VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce anomálií v lidském chování

Anomaly Detection in Human Behaviour

2020 Bc. Petr Dolejší

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Petr Dolejší**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Detekce anomálií v lidském chování Anomaly Detection in Human Behaviour
Jazyk vypracování:	čeština

Zásady pro vypracování:

Rozpoznávání lidských činností je významnou úlohou s rozmanitými aplikacemi. Úlohu lze dále specifikovat na rozpoznávání předem daných konkrétních činností nebo naopak na rozpoznávání (detekci) anomálií při vykonávání nějaké konkrétní činnosti. Ve své práci se zaměřte na druhou oblast. Detekovat anomálie můžete např. při chůzi, běhu, při chování řidiče, případně u jiné zvolené činnosti.

1. K detekci použijte příznaků odvozených z kostry člověka; kostru můžete detekovat pomocí software OpenPose.
2. Experimentujte s variantou 2D/3D detekce a s různými příznaky.
3. Detekci anomálií proveďte s využitím neuronových sítí; můžete použít např. síť LSTM. (Využijte některé z dostupných implementací konkrétní zvolené sítě, např. TensorFlow)
4. Funkčnost realizovaného řešení dostatečně ověřte.
5. Řešení a dosažené výsledky popište v textové části práce.

Seznam doporučené odborné literatury:

[1] Zhe Cao, Tomas Simon, Shih-En Wei, Yaser Sheikh: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, CVPR 2017
[2] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, Puneet Agarwal: Long Short Term Memory Networks forAnomaly Detection in Time Series, ESANN 2015

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.	prof. Ing. Pavel Brandštetter, CSc.
vedoucí katedry	*děkan fakulty*

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 15. května 2020 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. května 2020 .....

Rád bych na tomto místě poděkoval mé rodině za podporu při studiu a vedoucímu této práce za vedení a rady při vypracovávání této práce.

Abstrakt

Cílem této práce je vytvoření programu, který bude rozpoznávat anomálie v lidském chování z videa. Nejprve teoreticky je popisováno, jak se můžou rozpoznávat činnosti, rozdílný pohled na scénu u 2D a 3D, hloubkové mapy a popis fungování neuronových sítí.

V praktické části je popsána má práce, tedy rozpoznání póz ve videu, následná predikce a vyhodnocení predikce chování. Cílem je rozpoznat, kdy se jedná o anomálii a kdy se jedná o normální chování.

Klíčová slova: diplomová práce, OpenPose, neuronová síť, LSTM, anomálie, lidské chování

Abstract

The purpose of this thesis is to create an application, that will recognize anomalies in human behavior from video. In theoretical part, it describes how actions can be recognized, different views of the scene in 2D and 3D, depth maps and a description of the functioning of neural networks.

The practical part describes my work, ie the recognition of poses in the video, subsequent prediction and evaluation of behavior prediction. The goal is to recognize when it is an anomaly and when it is normal behavior.

Keywords: master thesis, OpenPose, neural networks, LSTM, anomaly, human behaviour

Obsah

Seznam použitých zkratek a symbolů

CPU	–	Central Processing Unit - procesor
CUDA	–	Compute Unified Device Architecture - paralelní výpočetní platforma umožňující stavět aplikace na GPU výpočtech
GPU	–	Graphics Processing Unit - grafická karta
LSTM	–	Long short-term memory - dlouze krátkodobá paměťová jednotka
OpenCV	–	Open source computer vision - knihovna pro počítačové vidění
RGB	–	Red Green Blue - červená zelená modrá
RNN	–	Recurrent neural network - rekurentí neuronová síť
TCN	–	Temporal Convolutional Network - temporální konvoluční sítě

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době, skoro na každém kroku můžeme vidět kolem sebe kamery. Data z nich se ale musejí nějak zpracovávat. Lidé nemůžu takové množství dat kolem sebe zpracovávat sami. Kamery a další zařízení v dnešní době již dokážou zpracovávat informace v obraze, ale vždy se jedná o zpracování jednoho snímku bez informace, co bylo na předchozích. Proto je potřeba rozpoznávat činnosti, tedy zpracovávat snímky, ale s ohledem, co již bylo.

Rozpoznávání lidských činností se může uplatnit v různých odvětvích, například sledování řidiče, hlídání prostoru, atd. Cíl této práce je rozpoznávání řidiče, zda nastane anomálie (zdravotní problémy - infarkt, epileptický záchvat, atd.) anebo je v pořádku. Využití rozpoznání anomálií je z důvodu, že pro trénování neuronové sítě se špatně shánějí nebo ani nejde sehnat učící data, např. pro infarkt. V této práci řeším, zda nastávají standardní činnosti, které neuronová síť správně predikuje anebo nestandardní situace, kde síť vyhodnotí něco jiného, tedy že rozdíl mezi predikovanou a reálnou hodnotu je veliký.

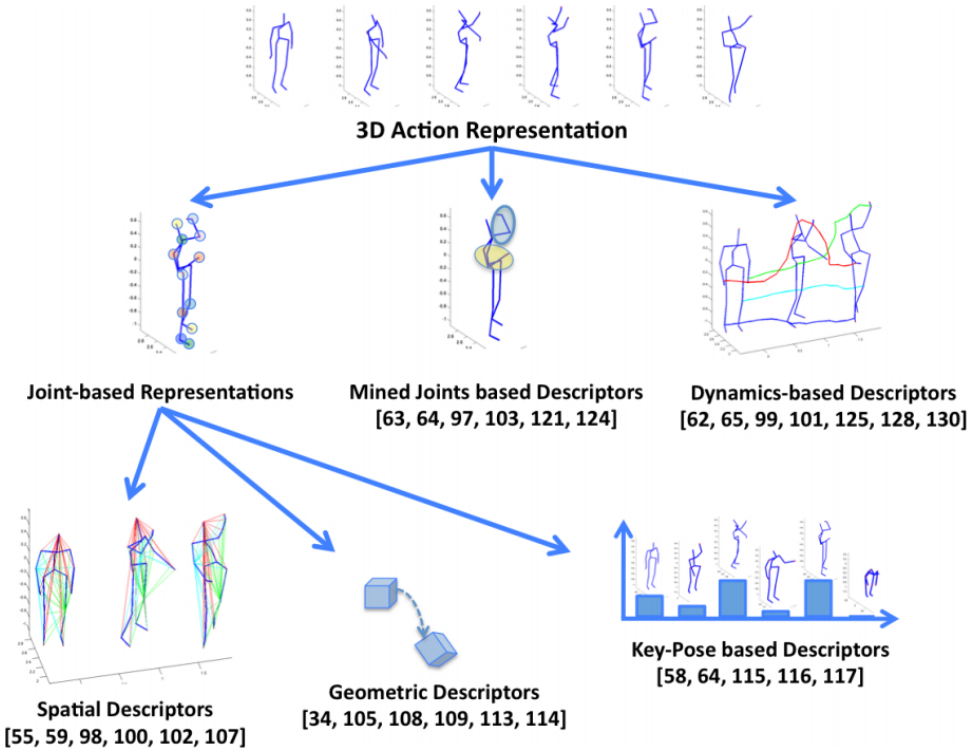
V rámci této práce jsem vytvořil trénovací a testovací videa, poté jsem vytvořil program pro nalezení a uložení příznaků pro osobu, která ve videu dělá různé činnosti a pak další program, který vyhodnocuje, zda se člověk chová normálně a nebo zda probíhá anomálie.

2 Metody rozpoznávání činností

Rozpoznávání akcí je proces, při kterém dochází k pojmenování akce prováděné člověkem na základě různých algoritmů. Vstupem do těchto algoritmů, na základě, kterých dochází k pojmenování akcí, může být například video.

Při vývoji algoritmů a metod rozpoznávání akcí však narážíme na několik problémů. Tyto metody musí být ideálně univerzálně použitelné na jakoukoliv osobu, která akce provádí v jakémkoliv prostředí. Musí být nezávislé na postavě člověka, jeho oblečení, věku, gestech, rychlosti provedení akce apod.

Metody rozpoznávání můžeme rozdělit na hlavní 3 kategorie, které si tady postupně rozebereme (převzato z [1]).



Obrázek 1: Grafické znázornění kategorizace metod rozpoznání činností. Převzato z [1].

2.1 Metody založené na kloubech

Tyto metody se snaží zachytit relativní polohy tělních kloubů. Ale i tuto část můžeme rozdělit na další 3 podkategorie [1].

První podkategorie zahrnuje přístup, kde se snažíme korelovat 3D tělesné klouby měřením všech možných párových vzdáleností (v daném čase nebo napříč časem) nebo jejich kovariančních matic. Druhá kategorie, tj. Geometrické deskriptory, zahrnuje metody, které se snaží odhadnout

posloupnost geometrických transformací potřebných k pohybu kostry v čase nebo k reprezentaci relativní geometrie podmnožin kloubů. Ve třetí kategorii, tj. Deskriptorech založených na klíčových pozicích, se vypočítá sada klíčových pozic a sekvence kostry je reprezentována z hlediska nejbližších klíčových pozic.

2.1.1 Prostorové deskriptory

Jedná se o korelaci míst kloubů a zohlednění všech párových vzdáleností 3D spojů. Tyto deskriptory postrádají jakékoli časové informace a mohou mít za následek nejednoznačné popisy akční sekvence. Jeden z přístupů, jak tento problém vyřešit je, že pozice těla je reprezentována zřetězením vzdáleností mezi všemi možnými páry spojů v aktuálním snímku, vzdáleností mezi spoji v aktuálním snímku a spoji v předchozím snímku, vzdáleností mezi spoji v aktuální snímku a v neutrální poloze (vypočteno zprůměrováním počátečních koster všech akčních sekvencí). Každá hodnota prvku byla seskupena do jedné z 5 skupin prostřednictvím k-means a byla nahrazena binárním vektorem pro reprezentaci každého indexu shluků. Odvozené pozice deskriptorů pro každou akci jsou určeny prostřednictvím rámce logistické regrese [1].

Podobným způsobem, se můžou využívat 3D polohy (a nikoliv vzdálenosti) kloubů ve stejné kostře mezi 3D klouby v aktuálním snímku a v předchozím snímku a mezi aktuálním snímkem a počátečním snímkem. Pro redukci rozměrů je použita analýza hlavních komponent (PCA) poskytující deskriptor [1].

2.1.2 Geometrické deskriptory

Metody v této kategorii se snaží představit si kostru pomocí geometrických vztahů mezi jednotlivými částmi těla. Například [1], geometrické znaky se skládají ze souboru booleovských znaků, z nichž každý je spojen se čtveřicí kloubů. Vzhledem k sadě čtyř bodů se tři z nich používají k identifikaci plochy. Čtvrtý prvek znázorňuje hodnotu 1, pokud je čtvrtý bod před rovinou, jinak předpokládá hodnotu 0. Tento typ prvku umožňuje reprezentovat geometrické vztahy mezi množinami kloubů a je odolný vůči prostorovým variacím, jako například velikosti kostry.

2.1.3 Deskriptory založené na klíčových pozicích

Tato kategorie zahrnuje metody, které se naučí slovník klíčových pozic a představují sekvenci akce z hlediska těchto klíčových pozic. Jako příklad, jak se může použít tento přístup [1]. Jako základní metoda je použit Histogram pohybových slov, kde zřetězené 3D kloubové pozice (nebo jejich odpovídající znaky) jsou seskupeny do K rozlišujících pozic pomocí K-means. Pohybová slova jsou detekována přiřazením každé reprezentace kostry k nejbližší klíčové pozici.

2.2 Metody založené na rozpoznávání vhodných částí těla

Jedná se o metody, které se snaží naučit, jaké části těla jsou zapojeny a / nebo jsou užitečné k rozlišení mezi akcemi. I když každý jednotlivec může vykonávat stejnou akci s různým sty-

lem, obvykle požadované pohyby zahrnují podobné podmnožiny kloubů. Detekce aktivovaných podmnožin kloubů může napomoci rozlišovat mezi různými třídami akcí.

Například metoda, která se může použít [1], modeluje každý kloub podle polohy a rychlosti ve sférickém souřadnicovém systému a korelací mezi umístěním a rychlostí znázorněnou jako ortogonální vektor ke umístění a rychlosti. Akce je modelována jako množina histogramů, z nichž každý je vypočten v sekvenci na specifickém příznaku a kloubu. K zachycení časové struktury akce se používá časová pyramida. U jiné metody můžeme použít genetický algoritmus k výběru kloubů, které představují akční třídu.

2.3 Dynamicky založené deskriptory

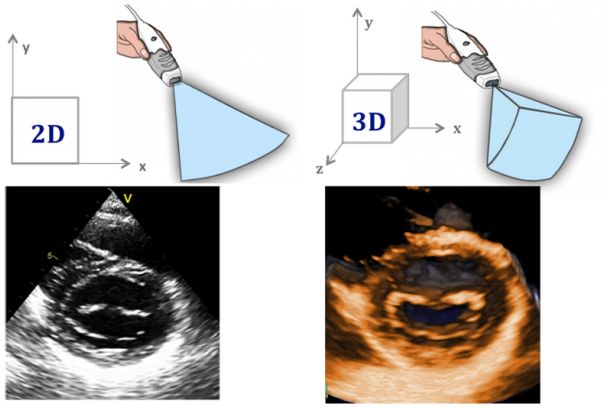
Metody pro 3D rozpoznávání kostry založené na této kategorii se zaměřují na modelování dynamiky podmnožin nebo všech kloubů v kostře. Toho lze dosáhnout zvážením například lineárních dynamických systémů (LDS), skrytých Markovových modelů (HMM) nebo smíšených přístupů [1].

Pro příklad, parametry získané z LDS modelování trajektorií kosterních kloubů popisují polohy a rychlosti jednotlivých kloubů. Kostra je rozdělena do několika částí těla reprezentovaných podmnožinami kloubů tak, aby reprezentoval celé tělo: horní část těla, dolní část těla, levá / pravá ruka, levé / pravé nohy atd. Z těchto částí těla se vypočítá reprezentace kontextu každé části těla. Histogram takového směru je pak normalizován. Sekvence kostry je reprezentována jako soubor časových řad (každou část těla jedna) jako jsou pozice a tečny kontextů. Časové řady jsou dále rozděleny do několika časových měřítek, které začínají od všech dat časové řady v určité sekvenci k menším a menším časovým částem stejné velikosti. Každá jednotlivá časová řada je modelována pomocí LDS metoda se naučí odpovídající systémové parametry. Odhadované parametry se používají k reprezentaci akce.

3 2D / 3D detekce

Při detekování pózy člověka, je vhodné zamyslet se, zda zvolit detekci v 2D nebo 3D. Každé zobrazení má svá specifika, kvůli kterých se program na to musí přizpůsobit. V této kapitole jsou popsány obě dvě zobrazení. Tyto rozdíly se ale týkají nejen detekce pózy člověka, ale v kterékoli jiné detekci, například ultrazvuk (obrázek 2).

2D je „plocha“, má pouze dva rozměry a pokud se plocha otočí na stranu, stane se čarou. 3D přidává hloubku. Tato třetí dimenze umožňuje rotaci a vizualizaci z více perspektiv. Je to v zásadě jako rozdíl mezi fotografií (2D) a sochami (3D).



Obrázek 2: Porovnání 2D a 3D snímání. Převzato z [2].

3.1 2D detekce

Pomocí 2D detekce můžeme detekovat pózu člověka v klasickém videu, které dokáže natočit například náš mobilní telefon. Vždy se jedná pouze o plochu. Tato plocha obsahuje pouze x a y souřadnice. Je tedy jednoduché data získat, ale problém může nastat, když kameru vůči naší scéně posuneme. Díky tomu by stejné příznaky byly v jiné pozici, než když kamera byla v původní pozici a mohlo by tedy dojít k nepřesnostem.

3.2 3D detekce

3D dokáže zachytit skutečná data povrchu předmětu a je méně narušena osvětlením a pozicí předmětu. Ve výsledku, detekce ve 3D vykazuje lepší výkon [3]. Tento druh detekce není tak jednoduchý, jako 2D detekce. Zde se již pracuje se třemi směry, tedy x, y a z. Pro získání vhodných data musíme využít speciální zařízení a nebo metody. Jeden ze způsobů může být převod 2D obrazu na 3D, ale to může být velmi náročný převod [4]. Další možností, jak získat data a pracovat s nimi, jak jsem to použil v této práci, je, že využívám 2D obraz, tedy složky x a y, a pro získání třetí složky z použiji hloubkovou mapu, kterou popisuji v následující kapitole

3.2.2. Díky transformace hloubkové mapy na RGB obraz (nebo na opak) mám k dispozici 3 složky pro každý bod v obraze.

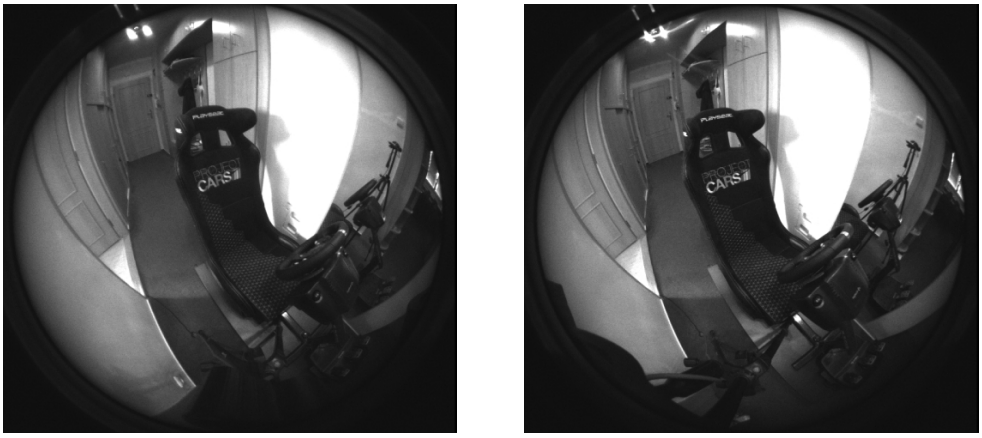
Dále krátce popíšu, jak se může 3D obraz získávat a zpracovávat.

3.2.1 Stereo kamery

Jak název napovídá, jedná se o systém více RGB snímačů, které ale musí být kalibrované tak, aby byl mezi nimi vypočítán 3D referenční systém. Je tedy potřeba vědět přesné pozice snímačů a vzdálenost snímačů od předmětu [5]. Použití více kamer zvýší přesnost výsledné mapy, ale zvýší náročnost výpočtů. Protože se ale používají RGB snímače, výsledná hloubková mapa může být ovlivněna prostředím, které snímá - například barevné splynutí předmětu s pozadím.

Na obrázku 3 jde vidět na stereo kameře mírný posun mezi oběma snímači. Snímače snímají stejnou scénu, ale každý z jiné pozice. Díky známé vzdálenosti mezi snímači lze vypočítat hloubkovou mapu.

V průběhu této práce jsem zkoušel jako jednu z možností, jak pracovat s 3D, použít stereo kameru (*Intel RealSense T265*), protože knihovna *OpenPose*, popsaná v kapitole 4.1 totiž obsahuje modul pro více kamer, ale v průběhu jsem se rozhodl od tohoto upustit a využívat kameru se strukturovaným světlem popsaná v 3.2.2.2.



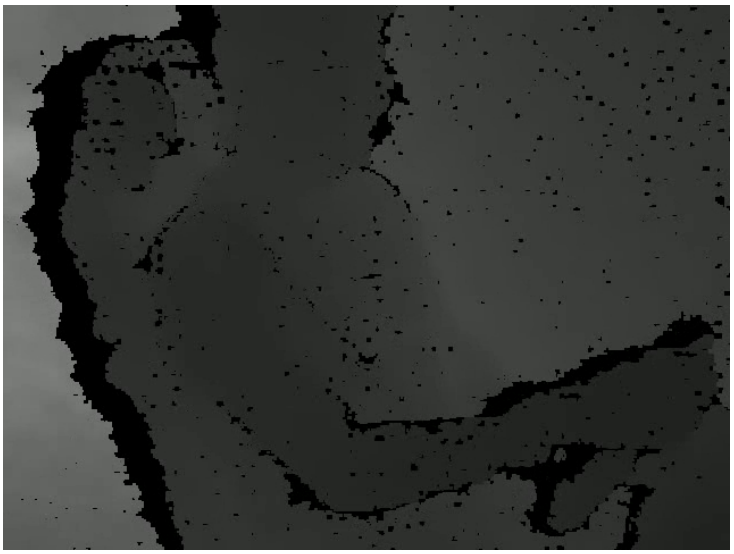
(a) (b)

Obrázek 3: Obraz z kamery Intel RealSense T265 - stereo kamera s dvěma senzory.

3.2.2 Hloubkové mapy

Hloubková mapa představuje obraz, kde každý bod představuje vzdálenost od kamery odpovídajícímu bodu ve scéně. Pro získávání těchto map tedy je potřeba jiná technologie, než klasické RGB snímače, které známe, například v mobilních telefonech. Existuje možnost jak využít i tyto RGB snímače, ale to za předpokladu několika splněných podmínek.

Doba vývoje těchto senzorů je oproti RGB senzorům malá, proto i kvalita zachycování hloubkových map není tak vysoká, jak jsme u klasických RGB obrazu zvyklí. Ale během posledních několika let zpět, proběhl v této oblasti velký posun dopředu.



Obrázek 4: Ukázka hloubkové mapy.

Člověk je chytrý a proto v obraze umí rozpoznávat předměty a lidi, ikdyž v obraze například oblečení člověka splývá s pozadím za ním. Proto mu stačí RGB obraz. Stroj ale takové možnosti nemá a proto stroji pro vyhodnocení hloubky nestačí RGB obraz. Pro strojové učení s obrazem je lepší využívat těchto hloubkových map. Textura ani barvy předmětů v obraze nemůžou ovlivnit výslednou hloubkovou mapu, protože se nesledují reálné barvy scény. Potíže můžou nastat u odrazivých a propustných materiálů.

Zde popíšu 2 nejznámější postupy, jak se hloubkové mapy získávají. Každý přístup k získání hloubkové mapy se hodí na jiné prostředí, proto nelze říct, že existuje univerzální řešení [4].

3.2.2.1 Time-of-Flight kamera Název této kamery se může přeložit "čas letu". Kamera tedy funguje na principu osvětlování scény modulovaným světlem a následném sledováním odraženého světla. Světlo má určitou rychlost, proto čím vzdálenější předmět je ve scéně, tím déle světlo bude putovat z kamery k předmětu, odrazí se a následně bude opět putovat zpět ke kameře. Jiný způsob, jak kamera může fungovat je, že se měří fázové rozdíly mezi vysílanými a přijímanými signály. Pro příklad, tuto technologii využívá *Microsoft Kinect* (2. verze). Na podobném principu funguje snímač LiDAR, který se pro tyto účely hodně využívá v automobilovém průmyslu [6].

Výsledná hloubková mapa může být nepřesná, pokud nastanou například vícenásobné odrazy, různé odchylky (osvětlovací, geometrické, radiometrické) a rozmazání pohybu [1].

3.2.2.2 Kamera se strukturovaným světlem

Tato kamera využívá principu, že do scény vyšle určitý vzor světla v neviditelné formě pro člověka (infračervené spektrum) - například tečky, čáry nebo jiné obrazce a pomocí senzoru snímá, jak se vyslaný vzor promítne do scény [1]. Tuto technologii využívá na příklad *Microsoft Kinect* (1. verze) a nebo *Intel RealSense D435*, kterou v této práci využívám.

Výsledná hloubková mapa může být nepřesná, pokud vzor, který kamera vysílá dopadne na reflexní a nebi průhledné povrchy.

(a) (b)

Obrázek 5: Obraz z kamery Intel RealSense D435 - (a) obraz v infračerveném spektru, (b) obraz v barevném spektru ve stejnou chvíli.

Jak je vidět v obrázku 5 v části (b), ikdyž nejsou světelné podmínky ideální, scéna je tedy docela tmavá, infračervené světlo, které kamera vysílá to neovlivní, takže i za horších světelných podmínek (nebo i špatných), touto technologií můžeme získat použitelnou hloubkovou mapu (a).

4 Nástroje

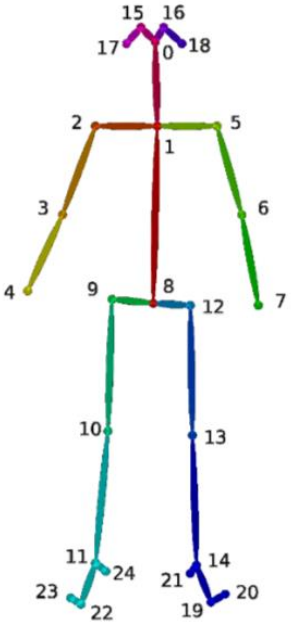
V této kapitole jsem popsal, které nástroje jsem používal a lehce naznačím, jak fungují.

4.1 OpenPose

Tato knihovna detekuje pózy v RGB obrazu. Vývojáři říkají, že se jedná o první knihovnu tohoto druhu [7]. Pokud je k tomu použit výkonný hardware, vývojáři knihovnu cílí jako real-time. Detekce může běžet synchronně i asynchronně. Knihovna podporuje detekci více osob v obraze, nicméně, já detekuji vždy pouze jednu osobu, více osob není žádoucí. Knihovna umí zpracovávat klasické 2D RGB obrazy, které jsou ve formě obrázků, videa a nebo stream z kamery. Podporované jsou systémy Windows, Linux i macOS. Výpočty můžou být spuštěny jak na CPU, tak i GPU (Cuda, OpenGL). Knihovna se neustále vyvíjí a přibývají nové funkcionality. Použití této knihovny bylo řečeno v rámci zadání.

Knihovna využívá konvoluční sítě pro detekci jednotlivých částí lidského těla, které následně přiřadí konkrétním osobám v obraze. Takto detekované a navzájem spojené body odpovídají hrubé stavbě lidského těla a jsou označovány jako póza osoby.

Pro detekci lze využít několik modelů těla. Pro detekci, jsem se rozhodl využít model BODY_25, který popisuje lidské tělo 25-ti body. V popisu jsou paže, tělo, nohy a hlava.



Obrázek 6: Model BODY_25. Převzato z [7].

Knihovna je založena na *OpenCV* a neuronové síti *Caffe*. *OpenPose* se skládá ze tří různých bloků:

(a) detekce těla + nohy

(b) detekce ruky

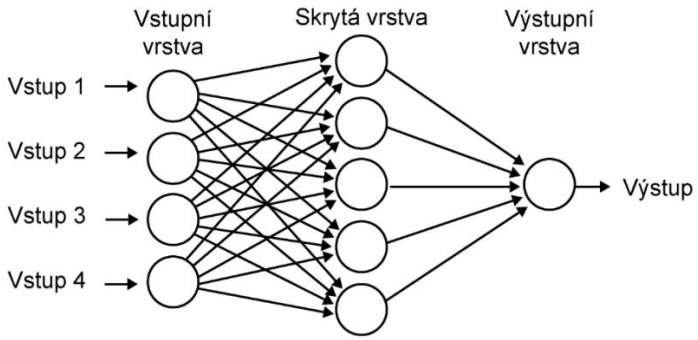
(c) detekce obličeje

Knihovna také obsahuje 3D detekci klíčových bodů v reálném čase v reálném čase, která je schopna předpovědět odhad 3D pozice z více synchronizovaných pohledů kamery.

4.2 Neuronové sítě

Neuronová síť je velmi populární a výkonná metoda, která se používá k modelování vztahu mezi vstupní proměnnou x a výstupní proměnnou y. Jako inspirace pro umělé neuronové sítě posloužil samotný lidský mozek. Fungování je velmi podobné.

Mozek sám od sebe neudělá nic, co se předem nenaučil, po dobu života se vyvíjí a učí a upravuje své chování podle zkušeností, které v životě nabyl. Neuronová síť se musí nejprve něco naučit, teprve poté může něco zpracovávat. Tak, jako se lidský mozek skládá z miliard buněk, které jsou navzájem různě propojeny a komunikují mezi sebou. Tyto buňky se nazývají neurony a společně tvoří neuronovou síť. Na základě naučení, tyto neurony různě reagují a v reakci skupiny neuronů můžeme dostat výsledek pro určitý vstup správný nebo očekávaný výsledek.



Obrázek 7: Uspořádání neuronů do vrstev v dopředné neuronové sítě. Převzato z [8].

Existuje mnoho typů neuronových sítí. V této práci jsem pro porovnání použil 2 typy neuronových sítí, které by se pro tuto problematiku mohly hodit – LSTM a TCN. V ranné části této práce jsem se zaměřil pouze na LSTM, protože TCN nevykazovala tak dobré výsledky, jako LSTM a navíc byla síť LTSM zadaná.

Při navrhování modelu sítě (tedy návrh konkrétní použití sítě) je mnoho možností, co a jak použít. Každý model má používá několik vrstev, jak je vidět na obrázku 7. Vždy jsou nejméně 2 vrstvy - vstupní a výstupní. Mezi nimi můžou být další - skryté. Vstupní vrstva má předem

daný počet vstupů a výstupní vrstva má předem daný počet výstupů. Ve vrstvách se nachází neurony, které jsou navzájem propojeny.

Při návrhu sítě se využívají další prvky nastavení [9].

loss funkce – tato funkce říká, jak se bude počítat rozdíl mezi vypočtenou a očekávanou hodnotou. V rámci experimentování jsem jich použil několik.

optimizer – protože neuronová síť má spoustu parametrů, tato funkce se na pozadí stará o to, aby vhodně tyto parametry nastavila tak, aby se chybovost klesala. Existuje více funkcí a každá může mít jinou strategii. V experimentování jsem některé použil.

aktivační funkce – někdy tak nazývaná přenosová funkce neuronu má na starosti modulaci neuronu v síti

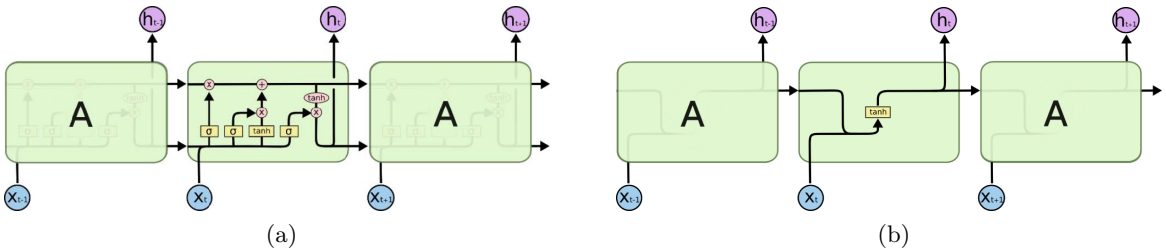
Při nastavování těchto funkcí nebo volby počtu vrstev jsou potřeba buď zkušenosti a nebo je potřeba experimentovat [10].

4.2.1 LSTM – Long Short Term Memory

Long Short Term Memory, jak je popsáno v [11] (a odkud zároveň čerpá tato kapitola) se jedná o speciální druh rekurentní sítě, která se umí naučit dlouhodobé závislosti. Díky její univerzálnosti se v současnosti jedná o jednou z nejpoužívanějších sítí. Použití této sítě bylo uvedeno v zadání této práce.

Tato síť byla specificky vytvořena pro to, aby se vyhnula problémům s dlouhodobými závislostmi, proto je jejich schopnost pamatovat si informace po dlouhé časové úseky prakticky zabudovaná v její struktuře, což usnadňuje práci a odpadá tedy složité učení. V tomto typu sítě nahrazujeme každý neuron tzv. LSTM buňkou, která obsahuje paměť.

Všechny rekurentní neuronové sítě mají podobu řetězce opakujících se modulů neuronové sítě. V běžných rekurentních sítích, tento opakující se modul mívá velmi jednoduchou strukturu. V LSTM místo jedné vrstvy neuronové sítě existují čtyři, které se vzájemně speciálně ovlivňují, což můžeme porovnat v obrázku 8.



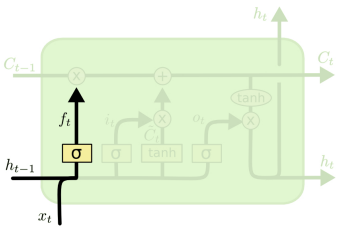
(a) (b)

Obrázek 8: Porovnání modulů LSTM (a) a běžné rekurentní sítě (b). Převzato z [11].

Základ LSTM je jednotkový stav buňky (cell state), což je vodorovná čára procházející na vrcholu buňky v obrázku 8 (a). Stav buňky je jako dopravní pás. Tento pás běží přes celý

diagram, v průběhu jsou menší lineární interakce. Pro informace je snadné, aby nebyly přes celou svou cestu vůbec změněny.

LSTM sítě mají schopnost přidávat a odebírat informace do nebo z cell state. Tato schopnost je však velmi opatrně regulována za pomocí bran (gates). Brány představují cestu jak volitelně povolovat informacím vstup do cell state. Skládají se z sigma vrstvy neuronové sítě a násobící operace, která pracuje bod po bodu.

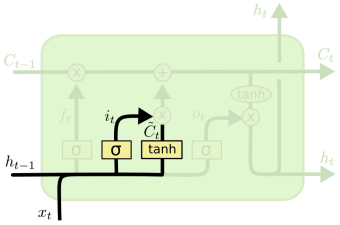


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Obrázek 9: Zapomínající vrstva pro odstraňování dat z aktuálního cell state. Převzato z [11].

Prvním krokem podle [11] je rozhodnout, jaké informace z buněčného stavu odhodíme. Výstup této vrstvy dává výsledek 1 (zcela nechat) a 0 (zcela se zbavit). Toto rozhodnutí je učiněno sigma vrstvou nazývanou forget gate layer (zapomínající vrstva). Tato vrstva se podívá na výstup z minulého modulu (h_{t-1}) a aktuální vstup (x_t) a vrátí číslo mezi 0 a 1 pro každé číslo v cell state (C_{t-1}), který se získal z minulého modulu. Toto číslo je v intervalu 0 až 1 a určuje, jak důležité je danou informaci si zapamatovat (funkce brány z minulé sekce).

Pro příklad si zvolme jazykový model, který se snaží předpovídat další slovo na základě předchozích. Někdy potřeba kontext je, a někdy ne. Když chceme předpovědět další slovo na základě kontextu předchozích, v takovém případě může buněčný stav zahrnovat rod současného předmětu, takže lze použít správná zájmena. Když uvidíme nový předmět, chceme zapomenout na rod starého předmětu.

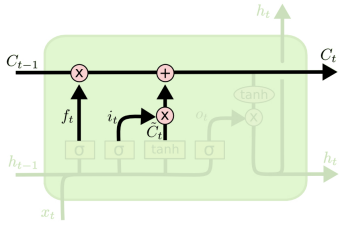


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Obrázek 10: Část modulu zabývající se tvorbou aktualizačních dat. Převzato z [11].

Dalším krokem v [11] je rozhodnout se, jaké nové informace uložíme do cell state. Toto se dělí na dvě části. První, sigma vrstva (input gate layer) určí, které hodnoty budeme aktualizovat. Poté druhá, tanh vrstva vytvoří vektor nových hodnot, který by mohl být přidán do cell state. V dalším kroku se předchozí 2 stavy zkombinují a vytvoří se aktualizace cell state. Aktualizace se následně provede.

V již zmíněném příkladu bychom chtěli do stavu buněk přidat rod nového předmětu, abychom nahradili ten starý, na který zapomínáme.

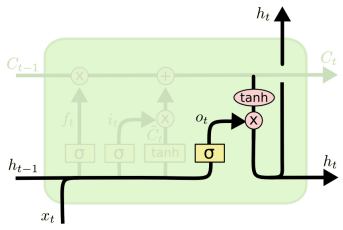


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Obrázek 11: Celá část LSTM modulu zabývající se aktualizací cell state. Převzato z [11].

Podle [11], dalším krokem je čas aktualizovat cell state staré buňky (C_{t-1}) a vytvořit z něho nový cell state (C_t). V předchozích krocích se už provedla rozhodnutí o datech, které budou aktualizovány, a nyní je tedy nutné provést tyto změny. Pokud se vynásobí minulý stav vektorem (f_t), způsobí to odstranění informací, které se již rozhodly zapomenout. Následně přičteme vektor ($\tilde{C}_t$), který obsahuje nové kandidátské hodnoty, jak moc jsme se rozhodli aktualizovat každou stavovou hodnotu. V tomto kroku se tedy provádí aktualizace cell state, a to jak odstraňování informací, tak jejich přidání

Pokud budeme pokračovat v předchozím příkladu, zde bychom vlastně upustili informace o rodu starého předmětu a přidali nové informace, jak jsme se rozhodli v předchozích krocích.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Obrázek 12: Část modulu, ve které probíhá tvorba výstupu a zasílání do dalšího modulu, popřípadě na jiný výstup. Převzato z [11].

Poslední krok v [11] je rozhodnutí, co bude na výstupu. Tento výstup bude založen na cell state, ale bude to filtrovaná verze. Sigma vrstva rozhodne, jaké části cell state budeme na výstupu. Pak jsme cell state proložíme tanh funkcí (abychom hodnoty měli mezi -1 a 1) a vynásobili jej výstupem sigma brány, takže na výstupu budou pouze ty hodnoty, které jsme se rozhodli.

Existuje více modifikací LSTM neuronových sítí, proto některé mohou fungovat na trochu jiném principu.

4.2.2 TCN - Temporal Convolutional Network

Tento druh sítě je docela nový. Jedná se o variantu konvoluční neuronové sítě -zvláštní případ klasických neuronových sítí, které slouží pro zpracování dat vykazujících určitou strukturovanost [12].

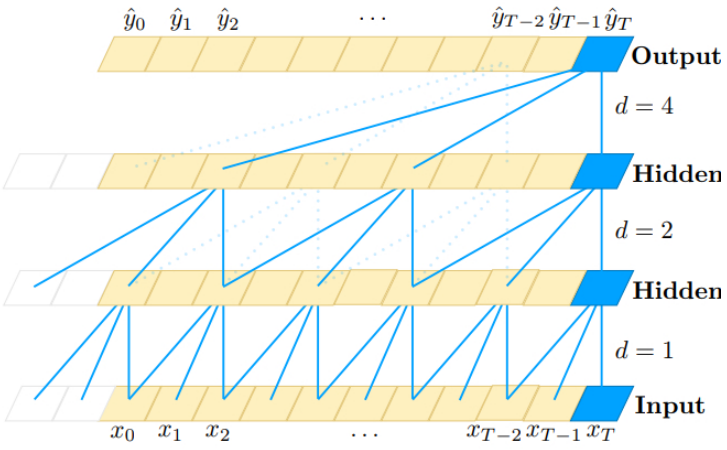
Architektura může mít sekvenci libovolné délky a mapovat ji na výstupní sekvenci stejné délky, stejně jako u RNN. TCN používá 1D plně konvoluční síť (FCN) architekturu, kde každá skrytá vrstva má stejnou délku jako vstupní vrstva, a přidává se nulová výplň délky (velikost jádra - 1) pro udržení následných vrstev stejnou délku jako předchozí.

Konvoluce v architektuře jsou příčinné, což znamená, že neexistuje žádná informace „úniku“ z budoucnosti do minulosti. TCN používá kauzální konvoluce, kde výstup v čase t je konvolován pouze s prvky z času t a prvky v předchozí vrstvě.

Jednoduše řečeno: TCN = 1D FCN + kauzální konvoluce

Nejdůležitější složkou TCN je rozšířená kauzální konvoluce. „Kauzální“ jednoduše znamená, že filtr v časovém kroku t může vidět pouze vstupy, které nejsou pozdější než t. Účelem kauzální konvoluce je dosáhnout většího receptivního pole (označujeme oblast ve vstupní vrstvě tvořenou buňkami, které mohou ovlivnit výstupní hodnotu této buňky) s méně parametry a méně vrstvami.

Tato základní architektura se může ohlédnout do historie s velikostí lineární v hloubce sítě a velikosti filtru. Zachycení dlouhodobých závislostí se tak stává opravdu náročným. Jedním z jednoduchých řešení k překonání tohoto problému je použití rozšířených konvolucí.



Obrázek 13: Kauzální konvoluce s dilatačními faktory d = 1, 2, 4 a velikostí filtru k = 3. Převzato z [12].

Výhod použití TCN pro sekvenční modelování je více. Na rozdíl od RNN (rekurzivní neuronové sítě), kde předpovědi pro pozdější časové kroky musí čekat na dokončení svých předchůdců, lze konvoluce provádět paralelně, protože stejný filtr se používá v každé vrstvě. Proto, jak v tréninku, tak v hodnocení, může být dlouhá vstupní sekvence zpracována jako celek v TCN namísto sekvenčně jako v RNN.

TCN může měnit svou velikost receptivního pole několika způsoby. Například stohování více rozšířených (kauzálních) konvolučních vrstev pomocí větších dilatačních faktorů nebo zvýšení velikosti filtru jsou všechny životaschopné možnosti (s možnými odlišnými interpretacemi). TCN tak umožňují lepší kontrolu velikosti paměti modelu a snadno se přizpůsobují různým oblastem.

Zejména v případě dlouhé vstupní sekvence mohou LSTM sítě snadno využít spoustu paměti pro uložení částečných výsledků pro jejich více buněčné brány. V TCN jsou však filtry sdíleny přes vrstvu, přičemž cesta zpětného šíření závisí pouze na hloubce sítě. V praxi tedy předpokládáme, že RNN pravděpodobně použijí více paměti než TCN.

4.2.3 Keras

Knihovnu pro práci s neuronovou sítí jsem použil Keras [13]. Tato knihovna je napsaná v jazyce Python a poskytuje vysokoúrovňové rozhraní pro práci s neuronovými sítěmi. Cílem této knihovny je usnadnit a urychlit vývoj a experimenty, čehož se snaží docílit zaměřením na uživatelskou přívětivost, modularitu a snadnou rozšiřitelnost.

Knihovna sama o sobě pouze dodává jednotné rozhraní. Pro práci vyžaduje další framework, přes který pak dále funguje. V současnosti Keras je podporován 3 frameworky, se kterými může pracovat - Tensorflow, Microsoft Cognitive Toolkit (CNTK) a Theano.

4.2.4 Tensorflow

Pro práci s neuronovou sítí jsem zvolil framework *TensorFlow* [14]. Jedná se o open-source framework, který je určen pro numerické výpočty. Pro urychlení, síť může spouštět kód na grafických kartách nVidia, protože využívá platformu CUDA. Framework byl vyvinut týmem GoogleBrain od společnosti Google a původně byl určen pouze pro interní práci a výzkum v rámci společnosti. V roce 2015 byl vypuštěn na veřejnost pod licencí Apache 2.0. Framework v současnosti podporuje práci v jazycích C++, Python, Javascript, Java, Go a Swift. Použití tohoto frameworku bylo zadáno v rámci zadání této práce.

4.3 Intel RealSense D435

Pro natáčení videí jsem využíval zapůjčenou kameru ze školy (za což škole děkuji) *Intel RealSense D435*. Jedná se o hloubkovou kameru, tedy, že snímá hloubku obrazu - jak daleko jsou předměty od kamery. Kamera obsahuje IR projektor, kdy před sebe zobrazuje v IR spektru tečky. Pomocí dvou (stereo) snímačů se vyhodnocuje, v jaké pozici každý snímač tečky vidí. Tento typ je popsán v kapitole 3.2.2.2. Princip je velmi podobný lidskému vnímání prostoru. Poslední snímač, který na kameře najdeme je RGB snímání. Výstupy z kamery tedy jsou hloubková mapa, kterou jsem popsal v kapitole 3.2.2 a RGB obrázek.

Obrázek 14: Kamera Intel RealSense D435.

4.3.1 Intel RealSense SDK

Tato knihovna pochází přímo od výrobce kamer *RealSense D435* a *T265*, které jsem využíval. Knihovna umožňuje streamování hloubkové mapy, RGB obraz a poskytuje vnitřní a vnější kalibrační informace v závislosti na typu kamery [15]. V textu taky někdy popsaná jako *librealsense*, podle toho, jak se jmenuje repositář na GitHubu.

Pro práci s kamerou, využívám tuto knihovnu. Obsahuje spoustu příkladů pro konkretní kamery a umožňuje zajímavé funkce pro práci s kamerou. Například, protože nelze docílit, aby snímače byly ve stejné pozici, snímají tedy trochu jiný prostor. Díky knihovny jsem schopný zarovnat buď hloubkovou mapu na RGB obrázek a nebo naopak. Tuto funkci využívám pro to, abych měl pro každý bod v RGB obrázku přesně daný bod v hloubkové mapě - mám tedy 3 souřadnice.

Knihovna, ve verzi 2.0, podporuje různé platformy a různé programovací jazyky, například Python, C++ a nebo C#.

4.4 OpenCV

Okrajově využívám v práci open-source knihovnu OpenCV [16], zároveň je součástí knihovny *OpenPose*. Tato knihovna obsahuje širokou škálu optimalizovaných funkcí zaměřených převážně na počítačové vidění, zpracování signálů a strojové učení. Knihovna je multiplatformní a je dostupná pro jazyky C++, Python a Java. Původně vznikla pod hlavičkou společnosti Intel, ale v současnosti spadá pod neziskovou organizaci OpenCV.org.

5 Moje řešení

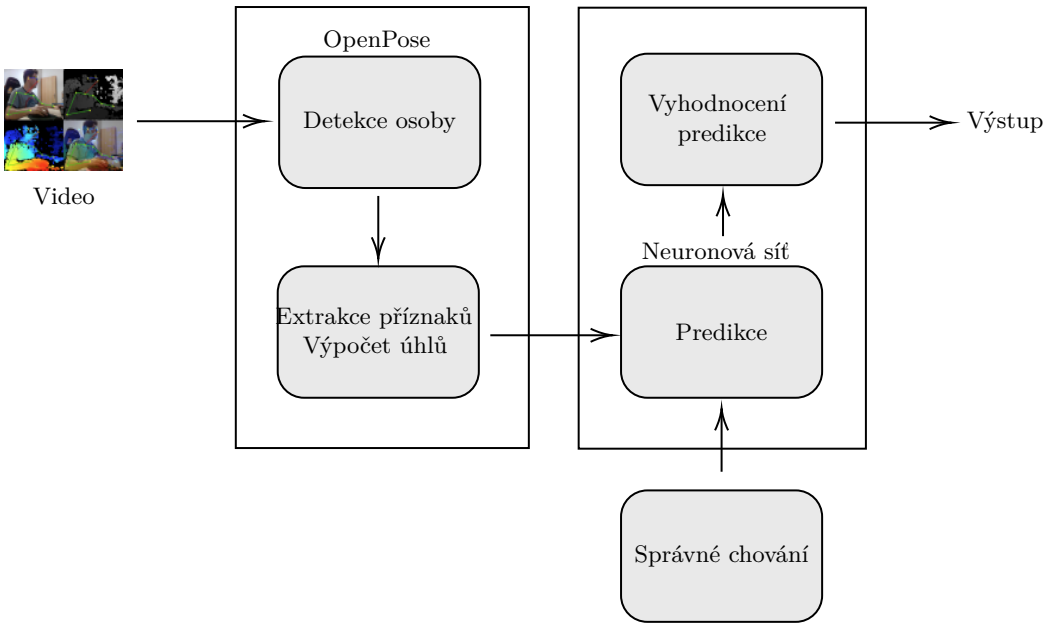
K vyřešení této práce existuje několik způsobů. Níže popíšu, které řešení jsem zvolil a jak funguje.

Práci bych rozdělil na dvě části. Protože, jako vstup je video buď ze souboru, anebo přímo z kamery v podobě streamu, je potřeba nejprve rozpoznat, zda ve videu se nachází člověk a případně, jakou pózu tam má. Pak je potřeba predikovat pózy a rozpoznávat anomálie, což je cílem této práce. Vstupem do těchto algoritmů, na základě, kterých dochází k pojmenování akcí, může být například video.

Při vývoji algoritmů a metod rozpoznávání akcí však narážíme na několik problémů. Tyto metody musí být ideálně univerzálně použitelné na jakoukoliv osobu, která akce provádí v jakémkoliv prostředí. Musí být nezávislé na postavě člověka, jeho oblečení, věku, gestech, rychlosti provedení akce, apod.

5.1 Struktura aplikace

Celá aplikace se skládá z dvou programů, dvou částí, programů. Na obrázku 15 jsem vytvořil schéma, jak calá aplikace funguje.



Obrázek 15: Schéma aplikace.

5.1.1 Rozpoznávání pózy člověka

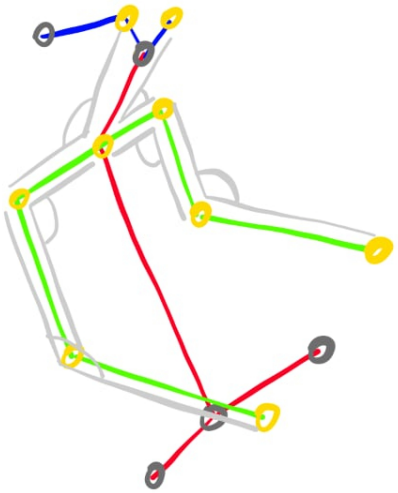
Pro člověka je jednoduché rozpoznat, zda předmět, který vidí, je člověk a jakou má pózu. Pro počítač to tak jednoduché není, protože vidí pole bodů. Proto je nejprve potřeba zjistit pózu člověka ve videu. Existuje několik způsobů rozpoznávání póz. Použil jsem rozpoznávání na základě

klíčových bodů na těle, protože byla zadána knihovna *OpenPose*, která tak funguje a kterou jsem již v popsal v kapitole 4.1.

Pro rozpoznání videa využívám video, které natáčím pomocí kamery *Intel RealSense D435*. Pokud využívám již natočené video, pomocí OpenCV si vyříznou potřebnou čtvrtinu videa, kterou vkládám do *OpenPose*, jak je popsáno v kapitole 5.1.1.

První program rozpoznává pózy člověka v obrazu. Z 25-ti dostupných bodů využívám pouze 10 bodů, na základě kterých vypočítám mezi úhel mezi zvolenými 3-mi body. Pro příklad, úhel loktu vypočítám z bodů zápěstí, loktu a ramene. Na základě pokusů jsem zvolil tyto kombinace (obrázek 16):

- Levý loket – levé rameno – krk
- Pravý loket – pravé rameno – krk
- Levé rameno – levý loket – levé zápěstí
- Pravé rameno – pravý loket – pravé zápěstí
- Levé rameno – krk – levé oko
- Pravé rameno – krk – pravé oko



Obrázek 16: Nakreslené úhly, které se vypočítávají.

Celkově takto spočítám 6 úhlů, protože u řidiče jsou tyto body nejviditelnější. Díky zvoleným částem těla a vypočítáním úhlů mezi nimi si mohu představit polohu rukou a náklon případně otočení hlavy. Souřadnice x a y získám z *OpenPose*. Protože mám napasovanou hloubkovou mapu na RGB video (senzory jsou mírně posunuty a nezabírají úplně stejnou scénu), snadno

tedy zjistím z souřadnici z hloubkové mapy. Výstup je textový soubor se sekvencí vektorů, které představují šest úhlů v jednotlivých snímcích.

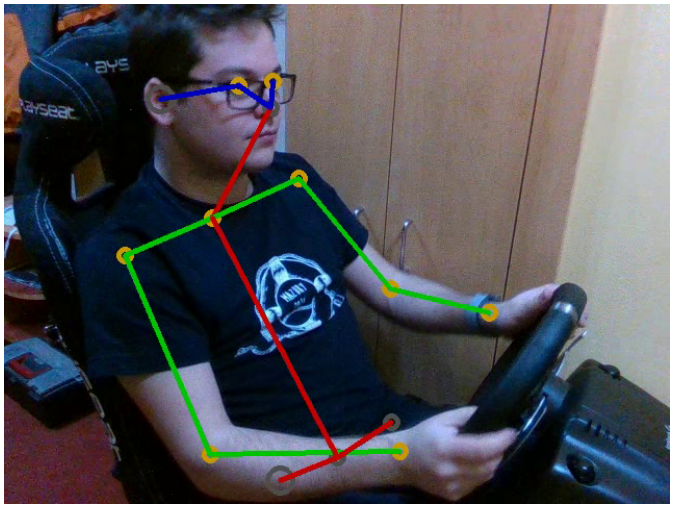
$$acos\left(\frac{\vec{BA}\cdot\vec{BC}}{\|\vec{BA}\|\cdot\|\vec{BC}\|}\right) \tag{1}$$

Výhoda je, že se jedná o úhel mezi 3D body, takže by nemělo programu vadit jiné natočení nebo umístění kamery vůči řidiči. Jako vedlejší výstup, který není potřebný pro další zpracování je video s vykreslenou kostrou člověka ve videu. Vykreslují se pouze ty příznaky, které *OpenPose* vyhodnotí s dostatečně velkou pravděpodobností. Do souboru *json.info*, který popisuji v kapitole 5.2.1, dopíšu ručně některé údaje na základě vizuálního posouzení videa. Pak mám testovací sekvence připravené pro další práci.

```
2.0056 2.0348 0.7107 1.5754 1.9471 0.6411
2.0399 2.5702 0.7489 1.3450 1.8223 0.8684
2.0453 2.6211 0.7815 1.3609 1.8374 0.8871
2.0526 2.1830 1.2190 1.5530 1.9677 0.9275
```

Výpis 1: Ukázka části souboru se zpracovanou sekvencí.

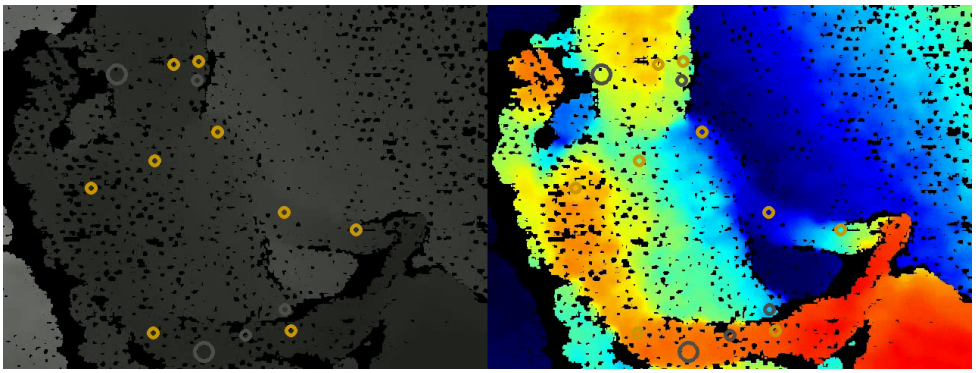
Zkoušel jsem zvolit různé kombinace příznaků (bodů na těle), ale tyto mi pro vyhodnocování chování řidiče vycházely jako nejlepší. Jsou nejlépe viditelné pro kameru. V kapitole 6.3.1 tyto pokusy popisuji.



Obrázek 17: Vykreslení kostry do obrázku.

Jak je vidět v obrázku [17], v obrázku je vykreslena kostra, kterou zpracoval *OpenPose*. Modré spojnice příznaků vykresluje hlavu, zelené vykreslují paže a červené trup. Jednotlivé příznaky jsou vykresleny pomocí kružnic, které mají rozdílnou barvu i průměr. Při zaměření na

barvu vidíme šedé a žluté. Šedé příznaky nevyužívám dále, žluté jsem zvolil pro další zpracování, tedy pro výpočet úhlů, jak jsem již popisoval výše. U průměrů kružnic, jsou použity 2 průměry - kružnice s malým průměrem, kde díky překrytí přímkou se zdá, že je kružnice plná a kružnici s větším průměrem. Menší průměr znamená, že jsem našel v hloubkové mapě nenulový bod - tedy že kamera našla hloubku. U většího průměru je opak, takže hloubková mapa je nulová a nevíme vzdálenost bodu od kamery.



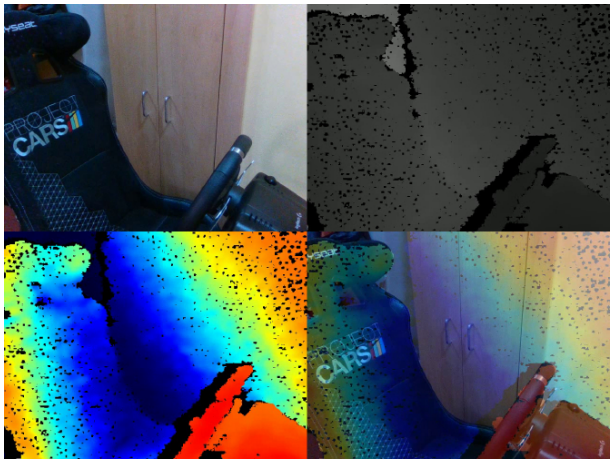
Obrázek 18: Hloubková mapa (černobílá a obarvená).

Jak můžeme vidět na obrázku [18], hloubková mapa není jednolitá, myslím tím, že se vyskytují v mapě černá místa, tedy že kamera v tomto bodě neví vzdálenost od ní. Pokud při výpočtu úhlů nastane tato situace, tedy že příznak z *OpenPose* se nachází v místě, kde je hloubková mapa nulová, zvolil jsem následující postup, jak tuto hodnotu vypočítat. Jsem si vědom, že tento postup není optimální, ale na základě pozorování, kamera nebo knihovna od kamery tyto místa umí dopočítávat. V programu *realsense-viewer*, což je program z knihovny *RealSense*, se nastavují různé parametry kamery. Když jsem si zapnul funkci dopočítávání, na základě mého pozorování jsem vyhodnotil, že tuto funkci nebudu používat. Nešlo nastavit nějaký limit, jak moc velké oblasti se budou dopočítávat, takže v obraze mohly vzniknout velké jednolité oblasti. Rozhodl jsem se, že využiji jiný postup. A ten je zcela jednoduchý. Vypočítám průměrnou hodnotu na základě okolí bodu a tu použiji jako hloubku. Uvědomuji si, že tento postup není přesný a pokročilý, ale pokud jsem potřebný bod potřeboval, dopočítat, tak v okolí bodu nikdy nebyla nějaká hrana, bod se tedy nacházel na rovině, proto jsem využil tento jednoduchý postup. Díky tomu můžu dopočítat pouze malé neznámé místa, ale pokud se již jedná o velkou plochu s neznámou hloubkou, tak hloubka pro konkrétní místo je neznámé.

Výsledné video vypadá jako obrázek [19]. Každý snímek ve videu se skládá ze 4 částí.

- 1. kvadrant - černobílá hloubková mapa
- 2. kvadrant - RGB video
- 3. kvadrant - obarvená hloubková mapa (lepe jde opticky rozpoznat vzdálenost)
- 4. kvadrant - překrytí RGB videa s obarvenou hloubkovou mapou

Při následném zpracování jsou potřeba pouze první 2 kvadranty z videa, ale 3. a 4. kvadrant slouží pro lepší optickou kontrolu nebo představu.



Obrázek 19: Výsledný obrázek ve videu.

Program pracuje buď s již existujícím videem, které se již natočilo a nebo s živým datovým proudem z kamery. Pro natočení video se jsem si vytvořil program (*realsense-D435/capture*), který vytvoří video na základě streamů z kamery *RealSense D435*. Video se skládá z již popsaných snímků. Dříve jsem používal analyzování videa také program pro datový stream z kamery v reálném čase (*realsense-D435/analyse*), ale to jsem z praktických důvodů později nepoužíval.

5.1.2 Predikce a vyhodnocení

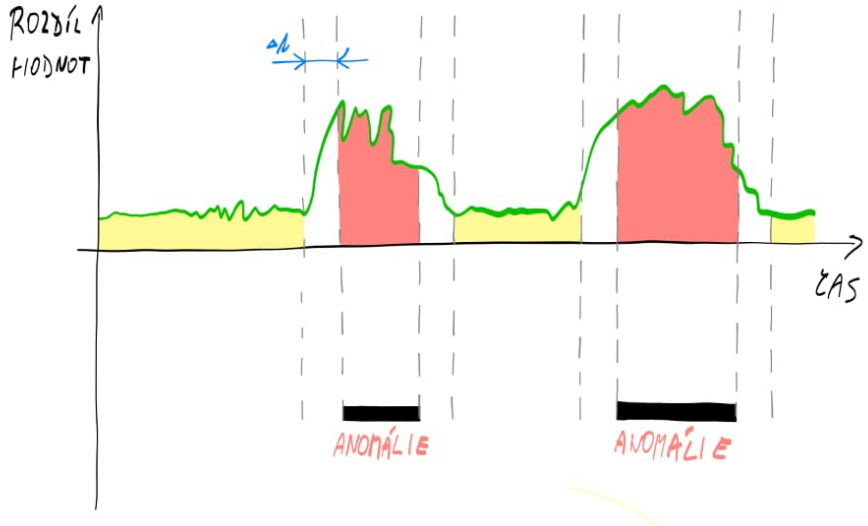
Proto, abych mohl vyhodnotit, zda se jedná o anomálii a nebo ne, využívám predikci a následné vyhodnocení. Jedná se o predikci, co nastane za určit počet snímků. Poté program vyhodnotí rozdíl mezi predikovanou a skutečnou hodnotou. Predikci vytváří model neuronové sítě, které jsem vytvářel. K tomuto se hodí konkrétně LSTM. Cíl predikce je, že síť se učí správné chování člověka, má tedy velkou vstupní sekvenci se správným chování. Na základě toho by naučený model měl správně predikovat časy, které nastanou. Pokud ale síť predikuje s velikým rozdílem hodnot (rozdíl mezi skutečnou a predikovanou hodnotou), predikce je špatná, a jedná se tedy o anomálii. Pokud se naučený model něco nenaučil, jedná se o anomálii.

Jako vstup do sítě mám v textovém souboru vektor úhlů, pro jednotlivé snímky z již zanalyzovaného videa. Síť má vytvořený naučený model se správnými činnostmi. Pro vytvoření modelu sítě používám stejnou strukturu trénovacích dat, jako pro testovací data.

Druhý program představuje neuronovou síť, kde do programu načtu soubor se sekvencemi úhlů, které pošlu do neuronové sítě. Ta jednotlivé sekvence vyhodnotí a poté vyhodnotím rozdíl mezi predikovanou hodnotou a reálnou hodnotou. Pokud je přiloženo video z předchozí části, vložím do videa ukazatel míry anomálie, který ukazuje, jak moc se liší mezi predikovanou hodnotou a reálnou hodnotou.

První skript (*actions/train.py*) trénuje model neuronové sítě. Po trénování sítě se vytvoří soubory, které jsou popsány později v kapitole 5.2.2. Pro nastavení sítě a dalších věcí je potřeba upravit kód skriptu.

Druhý skript (*actions/test.py*) pracuje se spouštěcím parametrem. Prvním parametrem definujeme spouštěcí ID modelu sítě, který najdu v přehledné tabulce modelů, která je popsána později v kapitole 5.2.2. Celé testování probíhá na všech testovacích sekvencích, kde se postupně jedna sekvence za druhou vyhodnocuje a pak se vypočítá celkové hodnocení.



Obrázek 20: Časový průběh a označení anomálií.

Hodnocení se vypočítává následovně.

1. Nejprve, jak jsem popsal v kapitole 5.1.1, dopíšu do souboru *info.json* místa, kde se nacházejí anomálie. Tento soubor má každá testovací sekvence zvlášť. Tento krok je sice vyžadován, ale pouze jen před prvním testováním, protože testovací program si potřebné údaje z něj přečte.

2. Pro každý čas predikce vypočítám rozdíl mezi skutečnou a predikovanou hodnotou a dostanu tedy mezi nimi rozdíl. Absolutní hodnoty rozdílů jednotlivých příznaků sečtu a mám tedy jednu hodnotu pro každý čas.

$$\begin{aligned} one\_time\_score \;=\; & |predicted\_value1[i] - actual\_value1[i]| + \\ & |predicted\_value2[i] - actual\_value2[i]| + \\ & |predicted\_value3[i] - actual\_value3[i]| + \\ & |predicted\_value4[i] - actual\_value4[i]| + \\ & |predicted\_value5[i] - actual\_value5[i]| + \\ & |predicted\_value6[i] - actual\_value6[i]| \end{aligned} \tag{2}$$

3. Každý čas predikce zařadím do jedné z kategorií a na základě s ní tak pracuji:
   (a) Anomálie – hodnotu přičtu k celkovému počtu rozdílových hodnot anomálií
   (b) Normální chování – tuto hodnotu přičtu k celkovému počtu rozdílových hodnot normálního chování
   (c) Interval mezi anomálií a normálním chování – tuto predikci nezapočítávám nikam, je tedy ve výpočtech přeskočena (Δ t)
4. Pokud mám rozděleny všechny časy a sečteny rozdílové hodnoty, vytvořím z nich průměr rozdílových hodnot anomálií a normálního chování.
5. Jako výsledek považuji poměr mezi těmito hodnotami. Snaha je, aby tento poměr byl co největší (rovnice 3).

$$test\_score = anomaly\_score / normal\_score \tag{3}$$

6. Pro konečný výsledek vypočítám průměr všech testovacích sekvencích, aby pro každý model jsem měl jedno číslo, které je porovnatelné mezi sebou (rovnice 4).

$$total = (first\_test + second\_test + third\_test)/3 \tag{4}$$

Program je rozdělen do dvou částí z několika důvodů. Hlavní důvod, proč jsem to takto udělal je, že oba programy běží na grafické kartě a jsou tedy náročné na výkon. Když pustím každou část samostatně, má grafická karta není tak rychlá, aby se jednalo o real-time zpracování. Další důvod, proč jsem udělal dva programy je, že *OpenPose* je C++ knihovna a práce s ní je také v C++ API. Knihovna sice má k dispozici Python API, ale v době začátku psaní ještě API nebylo vydané. Druhá část, tedy neuronová síť je řešena v jazyce Python.

5.2 Datová struktura

Abych se vyznal ve všech souborech, které tato práce má, vytvořil jsem si pro mně vhodnou adresářovou strukturu doplněnou o JSON soubory s důležitými informacemi, pro které jsem si následně vytvořil jednoduché skripty, které mi tyto informace přehledně zobrazí. Všechny uložené soubory se nachází ve složce *database*.

5.2.1 Sekvence

Sekvence mají svou strukturu. Jak už pro trénovací a nebo testovací sekvence, struktura je stejná. Nejprve mám prázdné (*train_empty* / *test_empty*), samostatně ve složce. Při vytváření sekvence pomocí *OpenPose* se vytvoří nová složka se stejným názvem v místě pro již zpracované sekvence (*train_openposed* / *test_openposed*).

info.json – soubor obsahuje ve formátu JSON informace ohledně analyzovaného videa, například: název, typ sekvence (testovací / trénovací), počet snímků (časů), délka analýzy a seznam anomálií. Tyto anomálie musím napsat manuálně na základě vizuálního vyhodnocení videa, ve kterých místech se vyskytuje anomálie. Každá anomálie má počáteční a konečné číslo snímku.

result.avi – video, které je zpracované pomocí *OpenPose* a je do něj zakreslena kostra, jak je popsáno v kapitole 5.1.1.

result_skeleton_coords.txt – textový soubor, který obsahuje spočítané úhly sekvence, který následně zpracovává neuronová síť. Pro další zpracování tento soubor není důležitý, ale dobře se v něm vizualizuje detekování pózy, případně se v něm provádí kontroly.

```
"name": "2_minutes_me_left",
"type": "test",
"elapsed_time": "0h 3m 21s",
"anomalies": [
    {
        "start": 640,
        "end": 700
    },
    {
        "start": 1100,
        "end": 1150
    }
]
```

Výpis 2: Ukázka části souboru json.info u testovací sekvence.

5.2.2 Modely

Natrénované modely se ukládají do složky *models*. Při trénování se vytvoří složka (např.: *30mins_50back_100epochs_LSTM_3*), kde v názvu jsou základní rysy modelu (trénovaný model, počet sledovaných snímků zpět, počet epoch a druh použití sítě - LSTM / TCN) a pořadové číslo, protože pro stejný název sítě může být jiné nastavení, například jiný počet vrstev, jiné použité funkce a další. Sítě, které jsem trénoval popisuji v kapitole 6.5. Každé trénování modelu neuronové sítě vytvoří ve složce několik souborů.

info.json – soubor obsahuje ve formátu JSON informace ohledně naučeného modelu sítě, například: typ sítě, počet epoch, počet vrstev, ID pro spuštění, počet sekvencí, které vyhodnocujeme zpět, názvy využitých funkcí. Jako další, doplňkové informace, soubor obsahuje jméno trénovací sekvence, datum začátku a konce trénování, délku tréninku, poslední hodnota loss funkce. Tento soubor se zpracovává při testování modelu a testovací sekvence a nebo při výpisu modelů.

loss.csv – historie hodnot loss funkce, pro každou epochu.

loss.png – vykreslení loss funkce v čase epoch.

model.py – uložený kód pro tento model.

model_plot.png – vykreslené schéma modelu.

trainded_model.h5 – uložený model v souboru typu HDF5.

Pro výpis všech modelů jsem vytvořil jednoduchý skript *db.py* (obrázek 21), který do přehledné tabulky v terminálu vypíše všechny vytvořené modely, na základě zmíněných složek a souborů *info.json*. Skript využívá vstupní parametry pro výpis menší a užší tabulky s méně sloupy (parametr *smaller*) a pak také setřídění řádků podle sloupců - podle čísla v závorkách u názvu sloupce v hlavičce tabulky. Při testování, pomocí parametru 2 z tabulky, se vybírá model, který se bude testovat. Další sloupce jsou pouze informativní, pro porovnávání.

```
Width of table (chars): 141
You can sort by cols by parameter with number in brackets
=============================================================================================================================================
| Name (1) - testing parameter (2)    | Net type (3) | Epochs (4) | Layers (5) | Steps back (6) | Activation (7) | Optimizer (8) | Loss (9) |
=============================================================================================================================================
| 30mins_50back_20epochs_LSTM_0 (1)   | LSTM         | 20         | 4          | 50             | relu           | adam          | mse      |
| 30mins_50back_20epochs_LSTM_1 (2)   | LSTM         | 20         | 4          | 50             | relu           | adadelta      | mse      |
| 30mins_50back_100epochs_LSTM_0 (3)  | LSTM         | 100        | 3          | 50             | relu           | adam          | mse      |
| 30mins_50back_100epochs_LSTM_1 (4)  | LSTM         | 100        | 4          | 50             | relu           | adam          | mse      |
| 30mins_50back_100epochs_LSTM_2 (5)  | LSTM         | 100        | 4          | 50             | relu           | adadelta      | mse      |
| 30mins_50back_100epochs_LSTM_3 (6)  | LSTM         | 100        | 3          | 50             | relu           | adadelta      | mse      |
| 30mins_50back_100epochs_LSTM_5 (7)  | LSTM         | 100        | 3          | 50             | linear         | adadelta      | mse      |
| 30mins_50back_100epochs_LSTM_4 (7)  | LSTM         | 100        | 3          | 50             | elu            | adadelta      | mse      |
| 30mins_50back_100epochs_LSTM_6 (8)  | LSTM         | 100        | 3          | 50             | relu           | sgd           | mse      |
| 30mins_50back_100epochs_TCN_0 (9)   | TCN          | 100        | 2          | 50             |                | adam          | mse      |
| 30mins_50back_250epochs_LSTM_0 (10) | LSTM         | 250        | 3          | 50             | relu           | adam          | mse      |
| 30mins_50back_250epochs_TCN_0 (11)  | TCN          | 250        | 2          | 50             |                | adam          | mse      |
| 30mins_50back_500epochs_TCN_0 (12)  | TCN          | 500        | 2          | 50             |                | adam          | mse      |
| 30mins_100back_50epochs_LSTM_0 (13) | LSTM         | 50         | 3          | 100            | relu           | adam          | mse      |
| 30mins_100back_50epochs_TCN_0 (14)  | TCN          | 50         | 2          | 100            |                | adam          | mse      |
=============================================================================================================================================
```

Obrázek 21: Náhled výpisu vytvořených modelů (obrázek neobsahuje všechny modely).

5.2.3 Testování

Při vyhodnocování testovací sekvence pro zvolený model se na konci testu také ukládají soubory. Pro tyto účely slouží složka *tests*. Testování probíhá vždy pro jeden naučený model neuronové sítě a pak pro všechny testovací sekvence.

info.json – soubor obsahuje ve formátu JSON informace ohledně provedených testů pro jednotlivé testovací sekvence. Pro každou sekvenci se vypíšou informace, jako finální hodnocení, hodnocení pro anomálie a normální chování, počty snímků pro každou kategorii a název testovací sekvence. Pro celkový přehled se ukládá délka testování, ID a název modelu.

differences_ *název testu* **.png** – pro každý test zvlášť se vykreslí rozdílová hodnota mezi skutečnou hodnotou a predikcí. V grafu jsou také zakresleny místa anomálií pro konkrétní test, zároveň jsou i vyznačeny meze přechodu mezi akcemi, které se ve výpočtu hodnocení vynechávají.

sequences_ *název testu* **.png** – pro konkrétní test se vykreslí do souboru hodnoty reálných příznaků (plná čára) a pro tyto příznaky predikce (čárkovaná čára). Barvy jsou vždy pro stejný příznak. V grafu jsou také zakresleny místa anomálií pro konkrétní test, zároveň jsou i vyznačeny meze přechodu mezi akcemi, které se ve výpočtu hodnocení vynechávají.

Pro přehledný výpis *json.info* souborů, vytvořil jsem ve složce skript *db.py*, který vypisuje všechny provedené testy. Program funguje na stejném principu, jako výpis modelů. Tabulka pro každý model obsahuje 5 hodnocení: celkové hodnocení, hodnocení prvního testovacího videa, hodnocení druhého testovacího videa, hodnocení třetího testovacího videa a na závěr hodnocení pro první a třetí testovací video dohromady, protože obě dvě videa mají kameru na stejné straně od řidiče. Tyto testy popisuji v kapitole 6.2.2.

Name (1)	Final score (2)	First test (3)	Second test (4)	Third test (5)	Left tests (6)
30mins_50back_100epochs_LSTM_18_0	1.005215712041	1.038490269590	0.914116382382	1.063040484153	1.050765376871
30mins_50back_100epochs_LSTM_12_0	1.260760851883	1.251450067460	1.282632531748	1.248199956440	1.249825011950
30mins_50back_100epochs_LSTM_11_0	1.593897350294	1.574827770406	1.494425149756	1.712439130721	1.643633450564
30mins_100back_50epochs_LSTM_0_0	1.939649249593	1.940513399988	1.717775263803	2.160659084989	2.050586242489
30mins_50back_250epochs_LSTM_1_0	1.994939753083	1.947414805294	1.751120806334	2.286283647622	2.116849226458
30mins_50back_100epochs_LSTM_1_0	1.996678756636	1.963849185607	1.725180829459	2.301006254844	2.132427720225
30mins_50back_20epochs_LSTM_0_0	1.996967784410	1.952382769251	1.742611015373	2.295909568606	2.124146168928
30mins_50back_100epochs_LSTM_10_0	1.999782133398	1.944590932149	1.741761329292	2.312994138752	2.128792535450
30mins_50back_100epochs_LSTM_6_0	2.047082462755	2.021699632221	1.854603250477	2.264944505567	2.143322068894
30mins_50back_100epochs_LSTM_8_0	2.072411281893	2.062927018989	1.779366424083	2.374940402607	2.218933710798
30mins_50back_100epochs_LSTM_19_0	2.122652847808	2.277884657326	1.575349093540	2.514724792559	2.396304724942
30mins_100back_100epochs_LSTM_0_0	2.135813738910	2.558033837850	1.806077364907	2.043330013972	2.300681925911
30mins_50back_250epochs_TCN_0_0	2.145367883020	2.741266294014	1.692647099592	2.002190255453	2.371728274733
30mins_50back_100epochs_LSTM_9_0	2.150083639318	1.902532006387	2.299362375708	2.248356535860	2.075444271123

Obrázek 22: Náhled výpisu otestovaných modelů (obrázek neobsahuje všechny testy).

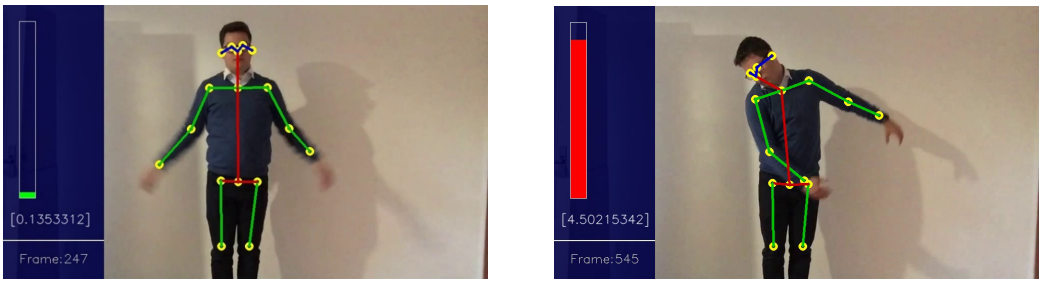
6 Experimenty

Vývoj a testování probíhalo na následující konfiguraci.

- CPU: 2x Intel Xeon E5-2630 2,3 GHz
- GPU: nVidia GeForce GTX 1060 6 GB
- RAM: 4x 8 GB, 1600MHz
- OS: Ubuntu 18.04
- SDD: Samsung 850 EVO 500 GB
- CUDA 10.2
- OpenPose 1.5.0
- TensorFlow 1.13.1
- Intel RealSense SDK 2.0

6.1 2D a 3D detekce

Nejprve jsem využíval 2D detekci. Pro zprovoznění celého procesu to bylo jednodušší. Jak jsem popisoval v kapitole 3, trénovací a testovací videa musela být ve stejné pozici a zachytávat stejnou scénu. Při změně pozice, jiném natočení již predikce nebyla správná. Při těchto testování jsem využíval videa nahraná tabletem před bílou zdí, jak jde vidět a obrázku 23. Pro vyhodnocování jsem využíval ukazatel míry anomálie.



(a) (b)

Obrázek 23: Detekce ve 2D - (a) normální chování, (b) anomálie.

Na již zmíněném obrázku, část (a) představuje normální chování, ukazatel tedy ukazuje nízké hodnoty, ale část (b) ukazuje anomálii (ruce jsou na jinou stranu, než hlava) a ukazatel tedy vysokou míru anomálie. Při tomto posuzování se těžko porovnávaly vytvořené modely (pouze vizuálně), takže jsem chtěl použít jiné porovnávání.

Poté jsem se rozhodl jít cestou 3D, pro zvýšení uplatnitelnosti. Kamera tedy nemusí být vždy při stejné pozici a ani sledovaná osoba nemusí být stejně. Při dalších pokusech jsem zkoušel snímání osoby při levostranném řízení a pravostranném řízení.

Nejprve jsem zkoušel kameru *Intel RealSense T265*, což je stereo kamera. Knihovna *OpenPose* obsahuje modul pro práci s více kamerami. Tato část se mi ale nepodařila zprovoznit se zmíněnou kamerou. Navíc podle repositáře na GitHubu jsem zjistil, že tento modul se dlouho neaktualizoval, takže vývojáři tuto část nerozvíjeli. Z toho jsem usoudil, že i z pohledu vývojářů tato cesta nemusí být ta nejlepší. Jako další pokus jsem využil kameru se strukturovaným světlem pro získání hloubkové mapy, popsanou v kapitole 3.2.2.2, pomocí které jsem vypracoval tuto práci.

6.2 Video

Pro nahrávání videí jsem použil kameru *Intel RealSense D435*, kterou jsem již popsal v kapitole 4.3. Pro simulování řidiče jsem si půjčil herní sedačku s volantem pro připojení k herní konzoli. Kameru jsem namontoval na stativ, aby simuloval připevnění kamery pod strop automobilu.

(a) (b)

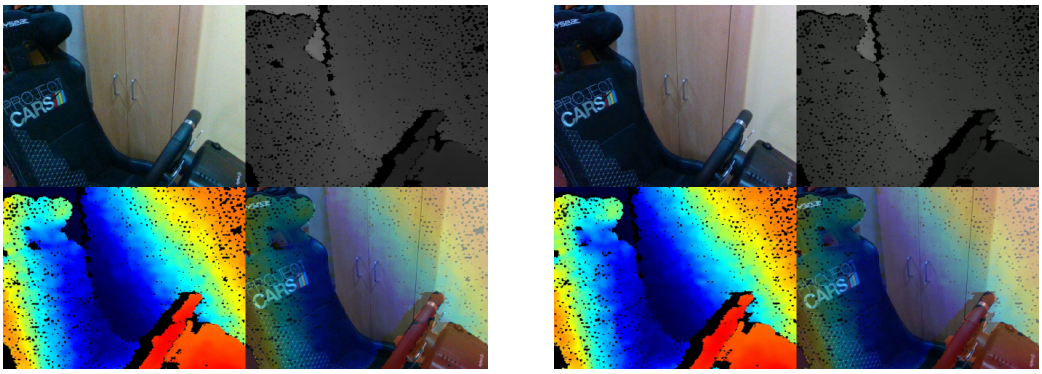
Obrázek 24: Nahrávání materiálů

Protože jsem nahrával velké množství dat, nejprve jsem zkusil, který kodek budu využívat pro nahrávání videí. Pro různé kodeky jsem nahrával stejný počet snímků, při snímání sedačky bez žádného pohybu - celé video je statický záběr. Můžu tedy říct, že otestování je naprosto spravedlivé. Výsledky jsem zapsal do tabulky 1.

kodek	fps	rozlišení	délka	bit rate	velikost souboru
X264	10	1280 x 960	30 sekund	0,7 Mb/s	24,5 MB
DIVX				17,1 Mb/s	61,5 MB
XVID				17,3 Mb/s	62 MB
WMV1				17,5 Mb/s	62,8 MB
WMV2				19,6 Mb/s	70,5 MB
MPEG				38,2 Mb/s	137 MB

Tabulka 1: Porovnání kodeků a velikosti videa.

Při porovnávání nejen velikosti pro 30 sekund, ale i kvality obrazu, MPEG má sice pohledově nejkvalitnější obraz, má také ale suverénně největší velikost. Na druhou stranu, X264 má nejnižší velikost, ale mám problémy tento kodek přehrát. Proto jsem se rozhodl, že pro rozumnou kvalitu a velikost budu používat XVID. Jedná se o open-source kodek, který má sice ztrátovou kompresi, ale jak vývojáři tvrdí, ztrácí se jen nedůležité detaily, které člověk nemůže vidět. Při pohledu na velikost souboru se jedná o méně než polovinu velikosti MPEG.



(a) MPEG (b) XVID

Obrázek 25: Porovnání kodeků.

6.2.1 Trénovací video

Pro trénování neuronové sítě jsem natočil video, které bylo 30 minut dlouhé. Do videa jsem se snažil zakomponovat různé prostředí a situace. Při natáčení jsem měl kameru na pravé straně.

Prostředí Jízda na dálnici (dlouhé dívání se dopředu a málo zatáčení), jízda ve městě (časté zatáčení, kruhové objezdy, časté se dívání kolem sebe - dávání přednosti druhým řidičům, čekání na semaforech, ..), jízda po serpentinách (ostré zatáčení), parkování (dívání se do bočních a zpětného zrcátka).

Situace Dívání se kolem silnice, dívání se do zrcátek, drbání se na obličeji a ruce, držení volantu více způsoby.

název	délka	osoba	pozice kamery
30_minutes	30 minut	já	vlevo

Tabulka 2: Trénovací video.

Obrázek 26: Snímek z trénovacího videa.

6.2.2 Testovací videa

Na testování modelů jsem si připravil tři testovací sekvence. Každá je v něčem jiná. Lišily se pozice kamery (popisovat pozici kamery budu čelem k autě) a také lidi. Vždy se jednalo o dvouminutové videa, které jsem následně zpracoval a dostal jsem již popsaný textový soubor, pomocí kterého jsem testoval modely neuronových sítí.

V každém testovacím videu jsou minimálně dvě anomálie, různě dlouhé i v jiných časech. V datovém souboru (*info.json*) jsem je popsal čísly snímků, jak je popsáno v kapitole 5.2.1.

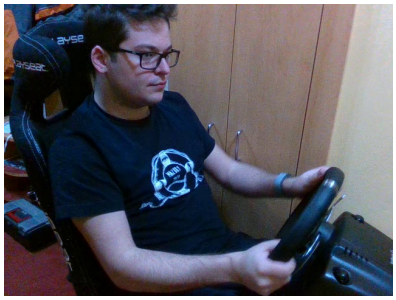
název	délka	osoba	pozice kamery	počet anomálií
2_minutes_me_left	2 minuty	já	vlevo	4
2_minutes_me_right		já	vpravo	3
2_minutes_milan_left		kamarád	vlevo	2

Tabulka 3: Testovací videa.

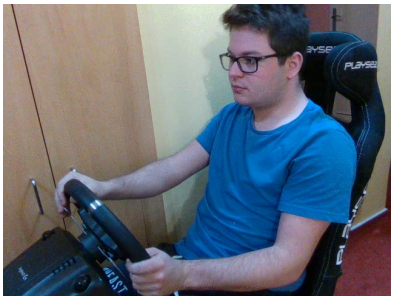
Obrázek 27 (a) ukazuje první testovací video. Ve videu jsem já a kameru mám na pravé straně. Při srovnání trénovacího videa a tohoto, kamera je mírně posunuta, tedy není úplně stejná.

V obrázku 27 (b), který představuje druhé testovací video jsem opět já, ale tentokrát mám kameru na levé straně.

U posledního, třetího testovacího videa, které je vidět na obrázku 27 (c), jako řidiče jsem natočil kamaráda a kameru měl na pravé straně. I v tomto případě, kamera není umístěna stejně tak, jako u předešlých videí s kamerou na pravé straně.



(a) První test



(b) Druhý test

(c) Třetí test

Obrázek 27: Testovací videa.

6.3 Detekce pózy ve videu

V rámci skriptu jsem použil některé body lidského těla a na základě nich jsem spočítal úhly mezi nimi. Tento proces jsem již popisoval v kapitole 5.1.1. Při analyzování se mi ale stávalo, že ve videu se objevovaly falešné osoby ("duchové"). Tento problém jsem vyřešil nastavením mezí pro délku paže mezi ramenem a loktem levé ruky. Tyto hodnoty jsem zkoušel nastavit tak, aby se hodily pro zvolenou scénu.

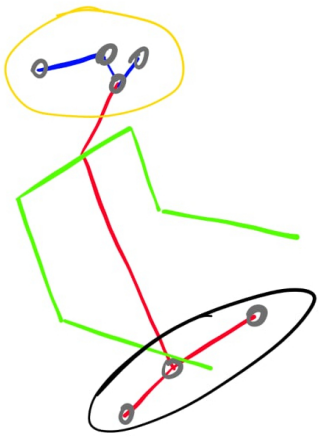
Pro zvolení správné hodnoty jsem si při celé analýze minimální a maximální délku zvolené části a při vyskytování nějakého "ducha"jsem určitou mez upravoval, aby byla vyšší, než nalezena minulá minimální hodnota. Při dodržování podobné vzdálenosti kamery od řidiče, program již nedetekoval žádné falešné osoby ve videu, tak ani nedetekoval žádné osoby. Vždy byla jen jedna, což je v tomto problému velmi žádoucí.

Testovací i trénovací videa jsem analyzoval pomocí *OpenPose*, které vytvořily soubor s úhly. Při vypočítávání úhlů se někdy stalo, že potřebný bod nebyl v obraze - buď *OpenPose* vyhodnotil nízkou pravděpodobnost konkrétního bodu a nebo prostě bod ve videu nebyl. V tom případě bylo řečeno, že počítaný úhel s tímto bodem je nula. Neuronová síť potřebovala mít nenulové hodnoty, proto jsem zvolil tento přístup.

Analyzování bylo rychlé, trénovací video (délka 30 minut) se analyzovalo 46 minut, trénovací videa (každé má délku 2 minuty) se analyzovala lehce přes 3 minuty. Je tedy zjevné, že má počítačová sestava nezvládá tento proces v reálném čase.

6.3.1 Volba příznaků

Při výběru volby příznaků, zkoušel jsem různé kombinace. Zaměřil jsem se na dvě oblasti, které jsem zakreslil od obrázku 28, oblast hlavy (žlutě zvýrazněno) a oblast pánve (černě zvýrazněno).



Obrázek 28: Oblasti testování volby příznaků.

Při pohledu na oblast pánve, lze vidět, že z pohledu kamery zde překáží většinou ruka. Z toho vyplývá, jak se mi v testech potvrdilo, že buď příznaky nešly vidět a nebo se vůbec nehýbaly, což je logické, protože řidič sedí na sedačce, a v těchto partiích se nehýbe.

Pokud se krátce zastavím u rukou, tam bylo nastavení jasné. Zkoušel jsem vypočítávat úhly tam, kde jsou i reálné klouby. Jiné pokusy jsem s rukami nedělal.

Nejvíce v této části experimentů jsem zkoušel pracovat s oblastí hlavy (žlutě zvýrazněno). Potřeboval jsem zajistit, abych věděl, v jaké pozici je hlava vůči tělu. Nejlepší kombinaci jsem měl pomocí kombinace oko - krk - rameno. Při použití uší jsem často neměl v záběru obě dvě uši. Pokud bych ale použil jen jedno ucho, nemohl bych naučený model používat při pravostranném i levostranném řízení.

6.4 Encoder-Decoder LSTM

V rámci experimentů jsem zkoušel zprovoznit model sítě s názvem Encoder-Decoder LSTM (nebo také autoenkodér). Tento model vysvětlím na základě porovnání s běžným modelem [17].

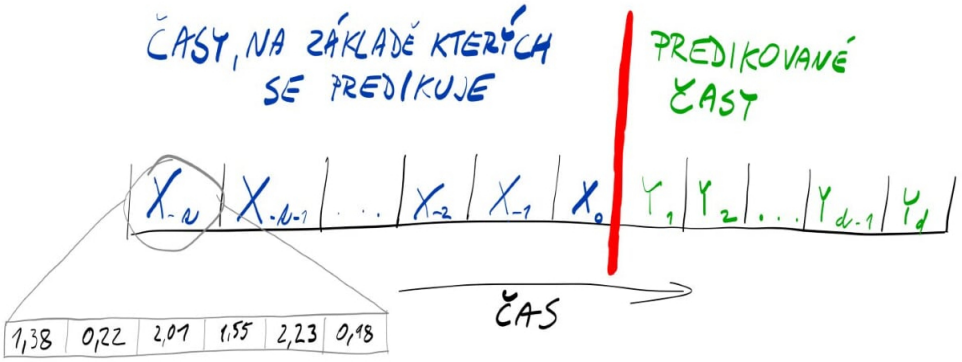
Běžně máme sekvence, kde pro jednu vstupní časovou hodnotu máme jednu časovou výstupní hodnotu (one-to-one) nebo pro více vstupních časových hodnot máme predikci pro jednu časovou hodnotu (many-to-one). Vždy tedy jako výsledek predikce dostaneme jednu předpovídanou hodnotu. Jako zmíněné hodnoty si můžeme představit pouze jednu hodnotu nebo celý vektor hodnot.

Model Encoder-Decoder, na rozdíl od předchozího příkladu, může predikovat více hodnot, tedy další sekvenci. Tento problém se někdy nazývá v angličtině sequence-to-sequence prediction problems, nebo zkráceně seq2seq [17].

Trénovací sekvence pro trénování modelu se musí vhodně připravit. Vstupní sekvenci transformuji do formátu vhodného pro tento problém. Zde jsou potřeba 2 pole. Pro každý čas v původní sekvenci (kromě několika prvních časů, protože pro ně nemám dostatečný počet časů před ním - nemám dostatečnou historii, na základě které může síť predikovat), do prvního pole vložím t časů zpět, tedy počet na základě kterých se predikuje. Do druhého pole se na konec vložím d hodnot, které by se měly na základě hodnot z prvního pole měly predikovat. Pomocí těchto dvou polí tedy říkám, co by se mělo predikovat (vzorec 6), když mám konkrétní vstupy (vzorec 5). Ukázky představují strukturu indexu v polích pro jeden čas.

$$[X_{-t}, X_{-t+1}, X_{-t+2}, ..., X_{-2}, X_{-1}, X_0] \tag{5}$$

$$[Y_1, Y_2, ..., Y_{d-1}, Y_d] \tag{6}$$



Obrázek 29: Časová osa predikce autoenkodéru.

Ukázku kódu ve výpisu 3 můžeme rozdělit na 2 hlavní části - kodér a dekodér. Kodér (anglicky Encoder) má na starosti, aby se síť naučila vztah mezi časy ve vstupní sekvenci a vytvořila se vnitřní reprezentace těchto časů. K implementaci modelu kodéru lze použít jednu nebo více

```
n_features = 6     # počet příznaků
n_steps = 50       # kolik snímků sledujeme pro predikci

model = Sequential()
# kodér
model.add(LSTM(32, activation='relu', input_shape=(n_steps, n_features)))

# spojení kodéru a dekodéru
model.add(RepeatVector(n_features))

# dekodér
model.add(LSTM(32, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(n_features)))

model.compile(optimizer='adam', loss='mse')
```

Výpis 3: Encoder-Decoder LSTM model

vrstev LSTM. Výstupem tohoto modelu je vektor pevné velikosti, který představuje vnitřní reprezentaci vstupní sekvence. Počet paměťových buněk v této vrstvě definuje délku tohoto vektoru s pevnou velikostí.

K implementaci modelu dekodéru lze také použít jednu nebo více vrstev LSTM. Tento model čte z výstupu s pevnou velikostí z modelu kodéru. Jako výstup pro síť používá Dense vrstva. Stejné váhy lze použít pro výstup každého časového kroku ve výstupní sekvenci obalením Dense vrstvy v TimeDistributed obalu.

TimeDistributed umožňuje opakovaně použít stejnou výstupní vrstvu pro každý prvek ve výstupní sekvenci. Poslední vrstva, která zároveň slouží jako výstupní vrstva, se nazývá Dense vrstva.

Aby jsme spojili kodér a dekodér dohromady, využívám vrstvu RepeatVector. Tato vrstva jednoduše opakuje zadaný 2D vstup vícekrát a vytvoří tak 3D výstup.

Pro srovnání jsem vytvořil několik modelů, které se liší pouze počtem předpovídaných časů do předu, ale nepovedlo se mi tyto modely otestovat. Proto dále pokračuji pouze s předpovídáním jednoho času.

6.5 Trénování modelů neuronové sítě

Trénovat neuronovou síť pro konkrétní použití není jednoduchá disciplína, protože nelze jednoznačně říct, jakou síť použít, jaké vrstvy použít a kolik jich má být, které funkce použít, počet epoch a nebo při predikci, na základě kolik snímků zpět síť se předpovídá, co nastane.

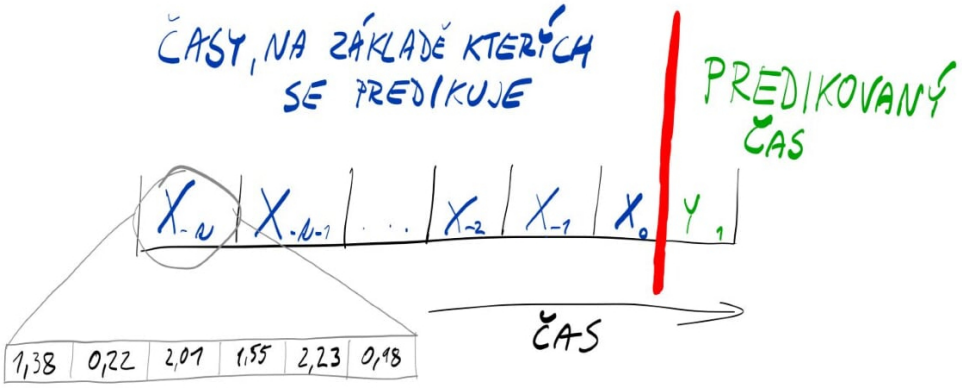
6.5.1 Metoda predikce

Základní metodou, kterou jsem u všech modelu použil je, že pro vstupních t časů zpět síť předpovídá jeden čas, který nastane. Jak jsem popisoval v kapitole 6.4, zkoušel jsem také predikovat více časů najednou, ale v mých modelech jsem predikoval pouze jeden.

Pokud mám výstupní sekvenci z *OpenPose*, tedy pro každý čas mám vektor šesti hodnot, musím tyto časy upravit do formátu, aby s nimi mohla síť pracovat. Při trénování sítě jsou potřeba dvě pole. První pole obsahuje časy a jejich hodnoty, na základě kterých se predikuje (vzorec 7). Do druhého pole se ke stejnému indexu vloží jeden čas s hodnotami, který by se měl na základě hodnot z prvního pole predikovat (vzorec 8). Díky tomu síť zjišťuje rozdíl mezi predikovanou hodnotou a hodnotou reálnou. Při každé epoše, tedy opakování trénovacího procesu, se vnitřní váhy neuronů upravují tak, aby zmíněný rozdíl byl co nejmenší.

$$[X_{-t}, X_{-t+1}, X_{-t+2}, ..., X_{-2}, X_{-1}, X_0] \tag{7}$$

$$[Y_1] \tag{8}$$



Obrázek 30: Časová osa predikce mých modelů.

Predikce neprobíhá od začátku vstupní sekvence, ale začíná se predikovat až *t+1* čas, aby se mohlo predikovat na základě t časů.

6.5.2 Nastavení modelů

Při experimentování jsem měnil různé parametry, jako počet vrstev, počet neuronů ve vrstvě, loss funkce, aktivační funkce, optimizer a další.

Počet epoch – Jeden průchod neuronové sítě všemi trénovacími daty se nazývá epocha. Přidáváním epoch se zvyšuje schopnost sítě modelovat sekvence, po určitém počtu epoch se ale přesnost sítě se může zhoršovat. Proto je potřeba zvolit správný počet epoch trénování a nebo zkoušet, kolik epoch je nejlepší pro konkrétní řešenou úlohu.

Počet kroků zpět pro predikci – Predikce se vykonává na základě určitého počtu kroků zpět. Toto jsem popisoval v kapitole 6.5.1.

Počet vrstev – Nejjednodušší část je znát počet vstupních a výstupních prvků a počet jejich neuronů. Každá síť má jednu vstupní vrstvu a jednu výstupní vrstvu. Počet neuronů ve vstupní vrstvě se rovná počtu vstupních proměnných ve zpracovávaných datech. Počet neuronů ve výstupní vrstvě se rovná počtu výstupů spojených s každým vstupem. Výzvou však je znát počet skrytých vrstev a jejich neuronů. Na internetu jsou návody, ale i v této oblastí funguje varianta pokus a omyl.

Počet neuronů ve vrstvě – Počet neuronů ve vrstvě je počet modulů za sebou v jedné vrstvě. Počet použitých neuronů není univerzální a proto se musí pro zvolený úkol vhodně zvolit. Pokud se zvolí nízký počet, síť nemá dostatečnou kapacitu k tomu, aby se správně naučila. V opačném případě, tedy že je vysoký počet neuronů, síť se učí zbytečně dlouho a navíc může nastat stav zvaný přeučení. To je stav, kdy se neuronová síť příliš přesně naučí množinu trénovacích dat a to včetně jejích náhodných chyb nebo šumu a ztrácí schopnost generalizace. Reálně takové sítě bývají velmi špatné [18].

Loss funkce – Loss (ztráta) není nic jiného než chyba predikce neuronové sítě. Metoda výpočtu ztráty se nazývá loss funkce. Loss funkce se používá pro výpočet gradientů. A gradienty se používají k aktualizaci vah neuronové sítě při jejím trénování. Příklady, které jsem používal: MSE (Mean Squared Error), Mean squared logarithmic error, Mean absolute percentage error a nebo Binary crossentropy.

Aktivační funkce – Aktivační funkce jsou matematické rovnice, které určují výstup neuronové sítě. Funkce je připojena ke každému neuronu v síti a určuje, zda by měla být aktivována („vystřelena“) nebo ne, na základě toho, zda je vstup každého neuronu relevantní pro predikci modelu. Dalším aspektem aktivačních funkcí je to, že musí být výpočetně efektivní, protože jsou vypočteny na tisíce nebo dokonce miliony neuronů pro každý vzorek dat. Moderní neuronové sítě používají techniku zvanou backpropagation pro trénink modelu, který klade zvýšený výpočetní tlak na aktivační funkci a její derivační funkci. Příklady, které jsem mimo jiné použil: Relu, Elu, Linear a nebo Softmax [19].

Optimizer – Spojuje loss funkci a parametry modelu, aktualizací modelu v reakci na výstup loss funkce. Zjednodušeně řečeno, optimalizátory tvarují a formují model do co nejpřesnější možné podoby. Loss funkce říká optimalizátoru, jestli se pohybuje správným nebo nesprávným směrem [20]. Pro příklad uvedu: Adam, Adadelta a nebo SGD.

6.5.3 Natrénované modely

Snažil jsem se vytvořit různé kombinace zmíněných možností, abych vyzkoušel, kterým směrem bych měl směřovat. Nevýhoda byla, že některá trénování běžela dlouho, takže nebyl dostatečný

prostor pro dlouhodobé pokusy navíc, když jsem pro tuto práci využíval jiný operační systém než běžně používám, takže jsem nemohl využívat čas u práce pro trénování, například během jiných hodin výuky a nebo během práce. Všechny natrénované modely jsou uvedeny v tabulce 6 a jejich vizualizace v kapitole A.2. V každém modelu se vyskytují LTSM vstva a Dense vrstva. Každá vrstva má vstup (input) a výstup (output). Ve vizualizacích lze tyto vstupy a výstupy vidět. Pokud tam jsou tři údaje, první údaj je *None*, značí časově neomezenou vstupní sekvenci, druhý údaj značí počet snímků, na základě kterých se predikuje a na závěr třetí číslo říká, kolik neuronů vrstva obsahuje. Pokud jsou u vrstvy pouze dva údaje, první má opět hodnotu *None* a druhé číslo značí počet neuronů.

Každý model má jinou strukturu, jako například jiný počet neuronů, jiný počet vrstev, atd. a nebo má jiný počet epoch učení. Některé mají stejný počet vrstev, ale můžou se u nich lišit jiné funkce, například optimalizační. Některé prvky, které jsem vypozoroval popisuji v následující kapitole.

6.5.4 Zhodnocení

Jak jsem již napsal, při trénování neuronových sítí jsem zkoušel různé nastavení, typy vrstev a nebo jsem použil různé funkce. Uvedu příklady, které jsem vyzkoušel.

Počet epoch – Tento údaj ovlivňuje konečný model. Při pohledu na modely *30mins_50back_100epochs_LSTM_0* a *30mins_50back_500epochs_LSTM_0*, jedná se o stejné nastavení modelů (např, počet a typy vrstev, počty neuronů, aktivační funkce, atd.), kromě počtu epoch. Konečné hodnocení vyplývá lépe pro model s více epochami, ale je již možné, že při tak velkém počtu epoch se již síť přeučila.

Počet kroků zpět pro predikci – Obecně, při zvýšení počtu sledovaných kroků se výrazně nezvýšilo hodnocení modelů.

Počet vrstev – Přidáním počtu vrstev se nezvýšilo výrazně hodnocení modelů, spíše se zvýšil čas trénování.

Loss funkce – Správné a nebo špatné nastavení může velmi ovlivnit výsledný model. Při porovnaní modelů *30mins_50back_100epochs_LSTM_18* a *30mins_50back_100epochs_LSTM_0*, liší se pouze použitá loss funkce a ve výsledném hodnocení jsou tyto modely každý na jiném konci tabulky.

Při vytváření modelů jsem většinou použil metodu pokusů s tím, že jsem porovnával a podle toho přizpůsoboval dobrému trendu. Nemám s neuronovými sítěmi takové zkušenosti, abych přesně mohl vyhodnotit, co je potřeba upravit tak, aby síť predikovala co nejlépe.

6.6 Testování modelů neuronové sítě

Pro testování vytvořených modelů neuronových sítí jsem vždy použil již popsané 3 testy (kapitola 6.2.2), aby výsledná hodnocení moha být porovnatelná. Při pohledu na tabulku 7, kde jsou vypsané všechny modely a testy k nim, tak lze vidět, že některé modely nepredikovaly správně, protože hodnocení všech částí se pohybovalo kolem hodnoty jedna. Cílem těchto hodnocení je, aby číslo bylo co největší. V tabulce 4 jsou vypsány průměrný hodnoty pro jednotlivé testy ze všech modelů.

	1. test	2. test	3. test	celkově
Průměr	2.31764	1.80786	2.77188	2.29913

Tabulka 4: Průměr výsledků pro jednotlivé testy.

6.6.1 Nejlépe ohodnocený model

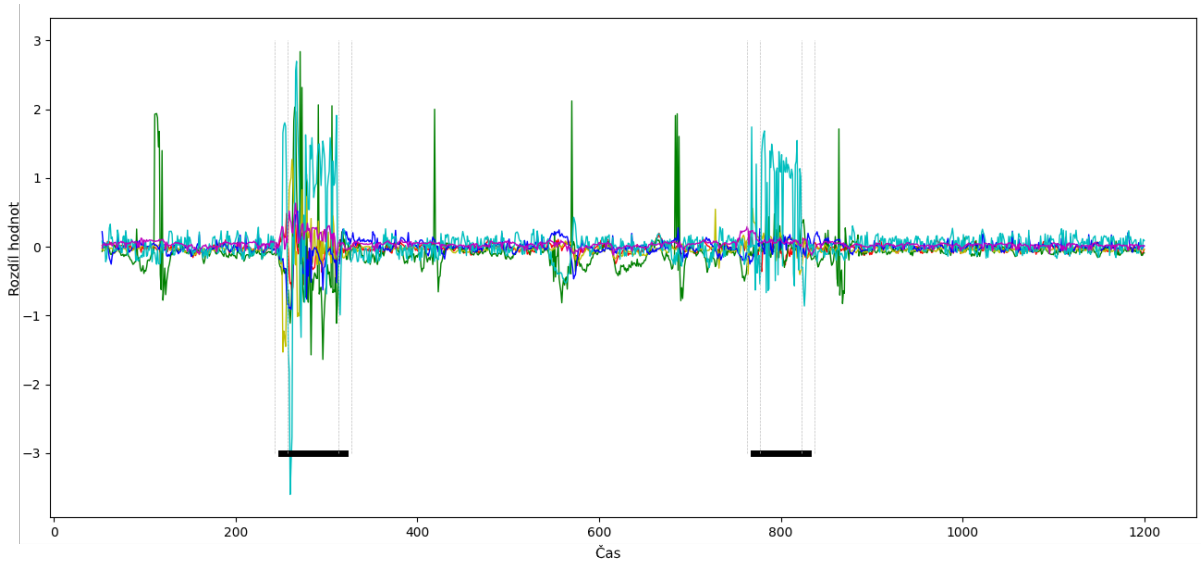
Ze všech modelů, model s názvem *30mins_50back_100epochs_LSTM_20* má nejlepší hodnocení. Pokud se ale podíváme do detailního hodnocení zobrazeného v tabulce 5, síť má velmi vysoké hodnocení těch testů, kde testovací video bylo natočeno ze stejné strany (1. test a 3. test). U 2. testu se hodnocení pohybuje v průměru těchto testů.

Název modelu	Hodnocení			
	1. test	2. test	3. test	celkově
30mins_50back_100epochs_LSTM_20	3.33237	1.75835	5.54894	3.54656

Tabulka 5: Výsledky testování.

Na obrázku 31 lze vidět vypočítané rozdíly hodnot mezi predikovanou a reálnou hodnotou pro jednotlivé časy. Cílem tohoto průběhu je, aby v anomálie, které jsou v grafu vykresleny černou vodorovnou čarou v hodnotě -3. Svislé šedé linky značí oblast, která se považuje za přechodem mezi anomálií a normálním chování, tedy, že se hodnoty v těchto oblastech nezapočítávají do hodnocení.

Podobně jako na předchozím obrázku, na obrázku 34 vidíme stejně zobrazené predikce. Rozdíl v těchto grafech je, že jsou zobrazeny konkrétní reálné hodnoty chování, které se detekovaly ve videu (plná čára) a predikovaná hodnota ve stejném čase (přerušovaná čára). Pro každý příznak je jiná barva s tím, že jsou vždy dvojice plná a přerušovaná čára ve stejné barvě pro jeden příznak. V tomto případě lze vidět, že až na několik míst neuronová síť predikovala správně, místy vyskočila zelená hodnota (pravý loket).



Obrázek 31: Rozdíl mezi predikovanými a reálnými hodnotami pro každý příznak.

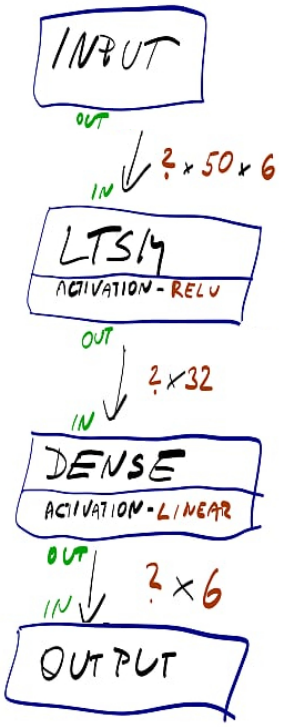
```
n_features = 6     # počet příznaků
n_steps = 50       # kolik snímků sledujeme pro predikci

model = Sequential()
model.add(LSTM(32, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(n_features))
model.compile(optimizer='adam', loss='mse')
```

Výpis 4: Kód sítě s nejlepším hodnocením.

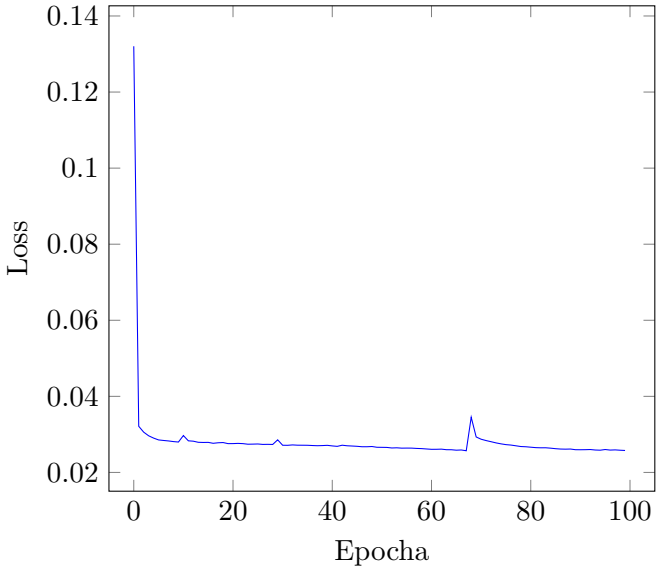
Pro tento model využívám LTSM vrstvu a jako výstupní vrstva slouží Dense vrstva, jak lze vidět na obrázku 32. LTSM vrstva má vstup 3D pole typu float, které má rozměry ? x 50 x 6. Otazník znamená, že síť očekává neznámý počet vstupních časů, který vkládáme k otestování modelu. Tento neznámý počet časů se týká všech ostatních vrstev. Druhý rozměr, tedy číslo 50 značí počet snímků, na základě kterého se predikuje jeden krok. Poslední rozměr, tedy v tomto případě 6 značí počet příznaků, se kterými pracujeme. Výstup této vrstvy již ve formátu 2D pole ve formátu ? x 32. Jinými slovy, pro každý predikovaný čas máme 32 neuronů.

Plně spojená vrstva, která následuje LSTM vrstvu a používá se pro výstup predikce, se nazývá Dense vrstva. Plně spojená vrstva znamená, že všechny neurony výstupní vrstvy jsou propojeny se všemi vstupními neurony. Vstupní ani výstupní neurony ale mezi sebou propojeny. Díky tomu se sníží počet neuronů na potřebný počet, v mém případě 6. Z této vrstvy máme tedy potřebný výsledek.

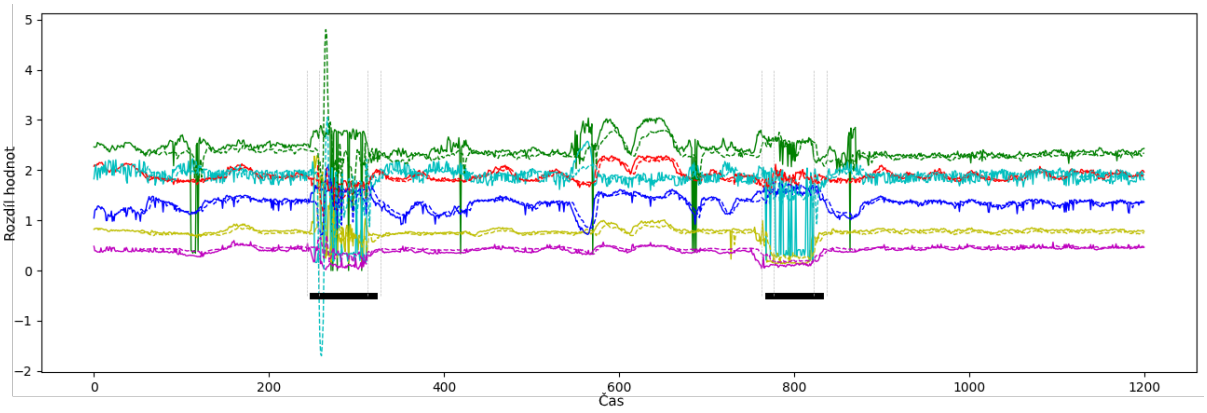
Výsledek nerunové sítě je sekvence vektorů, kde každý vektor představuje jeden čas ve vstupní sekvenci (ta, který byla jako výsledek ve *OpenPose*, tedy před transformací). Jak jsem již zmínil, výstupní sekvence bude mít o několik časů méně, než vstupní, protože predikce závisí na



Obrázek 32: Model sítě s nejlepším hodnocení



Obrázek 33: Vizualizace loss charakteristiky modelu s nejlepším hodnocení



Obrázek 34: Predikovaná (přerušovaná čára) a reálná hodnota (plná čára) pro příznaky.

prohlížení předchozích časů a proto tedy první predikovaný čas vůči vstupní sekvenci je pozice *dt - 1*, kde *dt* je počet snímků, které síť prohlíží zpět pro predikci. Výsledná sekvence se poté hodnotí. Pokud by vše bylo řešeno na dostatečně výkonném počítači a vše běželo v reálném čase, vypočítal by se rozdíl mezi predikovanými a reálnými hodnotami a na základě prahové hodnoty nebo jiného přístupu by se vyhodnocovaly konkretní anomálie a v případě řidiče by se mohla například zavolat záchranná služba.

7 Závěr

Na začátku této práce jsem popsal různé způsoby rozpoznávání činností. Při popisu a porovnávání 2D/3D detekce jsem také popsal hloubkovou mapu a její získávání. Poté jsem se přesunul k popisu mé práce. Nejprve jsem popsal nástroje, které jsem využíval. Další část obsahovala popis, jak s již popsanými nástroji pracuji a jak vypadá má aplikace. V poslední čísti jsem naznačil výsledek, nejlepší natrénovanou síť a ukázku dalších, které se mi povedly natrénovat.

Díky této práci jsem se naučil lépe pracovat s neuronovými sítěmi a pochopil jsem základy zpracování činností. Díky zapůjčení kamer od školy (*Intel RealSense D435* a *T265*) jsem si mohl vyzkoušet různé způsoby zpracování 3D obrazů.

Při zhodnocení tvorby modelů a jejich testování, vytvořil jsem o mnoho více modelů, než je zde uvedeno, protože se později ukázalo, například byly špatně natrénované. Bohužel se mi nepodařilo zprovoznit model enkodér-dekodér. Při pohledu na hodnocení modelů, hodnocení testů, které byly natočeny na stejné straně (tedy testovací i trénovací video) vykazovalo vyšší hodnocení, než když byly na rozdílné straně. Pokud tedy byly videa ze stejné strany, některé sítě predikovaly celkem správně a hodnocení bylo dobré (hodnoty větší než 3). Z rozdílných stran bylo hodnocení nízké, kde hodnoty se pohybovaly mezi hodnotami 1 a 2, což je níž, než předchozí případ. Je tedy lepší, používat videa ze stejné strany pro testování i trénování.

Odkazy

1. PRESTI, Liliana Lo; CASCIA, Marco La. 3D skeleton-based human action classification: A survey. *Pattern Recognition*. 2016, roč. 53, s. 130–147. ISSN 0031-3203. Dostupné z DOI: https://doi.org/10.1016/j.patcog.2015.11.019.
2. 2D, Comparison between; 3D. *https://www.escardio.org/Education/Practice-Tools/EACVI-toolboxes/3D-Echo/comparison-between-2d-and-3d* [online] [cit. 2020-05-02].
3. SAVRAN, Arman; SANKUR, Bülent; BILGE], M. [Taha. Comparative evaluation of 3D vs. 2D modality for automatic detection of facial action units. *Pattern Recognition*. 2012, roč. 45, č. 2, s. 767–782. ISSN 0031-3203. Dostupné z DOI: https://doi.org/10.1016/j.patcog.2011.07.022.
4. GHOBADI, Seyed Eghbal. *Real time object recognition and tracking using 2D/3D images*. 2010.
5. O' RIORDAN, Andrew; NEWE, Thomas; TOAL, Daniel; DOOLY, Gerard. Stereo Vision Sensing: Review of existing systems. In: 2018-12. Dostupné z DOI: 10.1109/ICSensT.2018.8603605.
6. ZHANG, Kai; XIE, Jiaxin; SNAVELY, Noah; CHEN, Qifeng. *Depth Sensing Beyond LiDAR Range*. 2020. Dostupné z arXiv: 2004.03048 [cs.CV].
7. *OpenPose: Real-time multi-person keypoint detection library for body, face, hands, and foot estimation* [online] [cit. 2020-02-15]. Dostupné z: https://github.com/CMU-Perceptual-Computing-Lab/openpose.
8. *Koncept umělé neuronové sítě* [online] [cit. 2020-01-18]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--uvod-do-neuronovych-siti--koncept-umele-neuronove-site.
9. *Neuronové sítě a rozeznávání obličeje – teoretické základy* [online] [cit. 2020-03-28]. Dostupné z: https://www.hobrasoft.cz/cs/blog/bravenec/neuronove-site-zaklad.
10. *Neuronové sítě* [online] [cit. 2020-03-29]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=21471.
11. *Understanding LSTM Networks* [online]. 2015 [cit. 2020-02-15]. Dostupné z: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.
12. *Keras Temporal Convolutional Network* [online] [cit. 2020-02-15]. Dostupné z: https://github.com/philipperemy/keras-tcn.
13. CHOLLET, François et al. *Keras* [https://keras.io]. 2015.

14. ASHISH AGARWAL et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Dostupné také z: `http://tensorflow.org/`. Software available from tensorflow.org.

15. *Intel® RealSense™ SDK 2.0* [online] [cit. 2020-02-18]. Dostupné z: `https://www.intelrealsense.com/sdk-2/`.

16. ITSEEZ. *Open Source Computer Vision Library* [`https://github.com/itseez/opencv`]. 2015.

17. *Encoder-Decoder Long Short-Term Memory Networks* [online] [cit. 2020-05-08]. Dostupné z: `https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/`.

18. KAČER, Petr. *Vícevrstvá neuronová síť*. 2013. Dostupné také z: `https://www.vutbr.cz/studenti/zav-prace/detail/65622`. Bakalářská práce. Vysoké učení technické v Brně.

19. *7 Types of Neural Network Activation Functions: How to Choose?* [online] [cit. 2020-05-02]. Dostupné z: `https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/`.

20. *Introduction to Optimizers* [online] [cit. 2020-05-02]. Dostupné z: `https://algorithmia.com/blog/introduction-to-optimizers`.

21. CAO, Zhe; HIDALGO, Gines; SIMON, Tomas; WEI, Shih-En; SHEIKH, Yaser. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *CoRR*. 2018, roč. abs/1812.08008. Dostupné také z: `http://arxiv.org/abs/1812.08008`.

22. *Intel® RealSense™ Depth Camera D435* [online] [cit. 2020-02-19]. Dostupné z: `https://www.intelrealsense.com/depth-camera-d435/`.

23. DRÁBEK, Pavel. *Sledování pohybu osob ve vymezeném prostoru [online]*. 2018. Dostupné také z: `https://theses.cz/id/qyzswy/`. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Ostrava. Vedoucí práce Tomáš FABIÁN.

24. *DivX v.s. XviD: How to Convert DivX/XviD Videos* [online] [cit. 2020-03-02]. Dostupné z: `https://www.videosolo.com/tutorials/divx-vs-xvid.html`.

25. *How to Develop LSTM Models for Time Series Forecasting* [online] [cit. 2019-10-07]. Dostupné z: `https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/`.

A Modely neuronových sítí

A.1 Tabulky

Název modelu	Vrstvy	LTSM	TCN	Dense	Activation	Optimizer	Loss
30mins_100back_100epochs_LSTM_0	4	3		1	relu	adam	mse
30mins_100back_50epochs_LSTM_0	3	2		1	relu	adam	mse
30mins_100back_50epochs_LSTM_1	3	2		1	relu	adam	mse
30mins_100back_50epochs_TCN_0	2		1	1		adam	mse
30mins_50back_100epochs_LSTM_0	3	2		1	relu	adam	mse
30mins_50back_100epochs_LSTM_1	4	3		1	relu	adam	mse
30mins_50back_100epochs_LSTM_10	3	2		1	linear	adam	mse
30mins_50back_100epochs_LSTM_11	3	2		1	linear	adam	binary_crossentropy
30mins_50back_100epochs_LSTM_12	3	2		1	relu	adam	binary_crossentropy
30mins_50back_100epochs_LSTM_13	3	2		1	relu	adam	mse
30mins_50back_100epochs_LSTM_14	4	3		1	relu	adam	mse
30mins_50back_100epochs_LSTM_15	3	2		1	relu	adam	mean_absolute_error
30mins_50back_100epochs_LSTM_16	3	2		1	relu	adam	mean_squared_logarithmic_error
30mins_50back_100epochs_LSTM_17	3	2		1	relu	adam	mean_absolute_percentage_error
30mins_50back_100epochs_LSTM_18	3	2		1	relu	adam	squared_hinge
30mins_50back_100epochs_LSTM_19	3	2		1	softmax	adam	mse
30mins_50back_100epochs_LSTM_2	4	3		1	relu	adadelta	mse
30mins_50back_100epochs_LSTM_20	2	1		1	relu	adam	mse
30mins_50back_100epochs_LSTM_3	3	2		1	relu	adadelta	mse
30mins_50back_100epochs_LSTM_4	3	2		1	elu	adadelta	mse
30mins_50back_100epochs_LSTM_5	3	2		1	linear	adadelta	mse
30mins_50back_100epochs_LSTM_6	3	2		1	relu	sgd	mse
30mins_50back_100epochs_LSTM_7	3	2		1	elu	sgd	mse
30mins_50back_100epochs_LSTM_8	3	2		1	elu	adam	mse
30mins_50back_100epochs_LSTM_9	3	2		1	linear	sgd	mse
30mins_50back_100epochs_TCN_0	2		1	1		adam	mse
30mins_50back_20epochs_LSTM_0	4	3		1	relu	adam	mse
30mins_50back_20epochs_LSTM_1	4	3		1	relu	adadelta	mse
30mins_50back_250epochs_LSTM_0	3	2		1	relu	adam	mse

Tabulka 6: Tabulka modelů

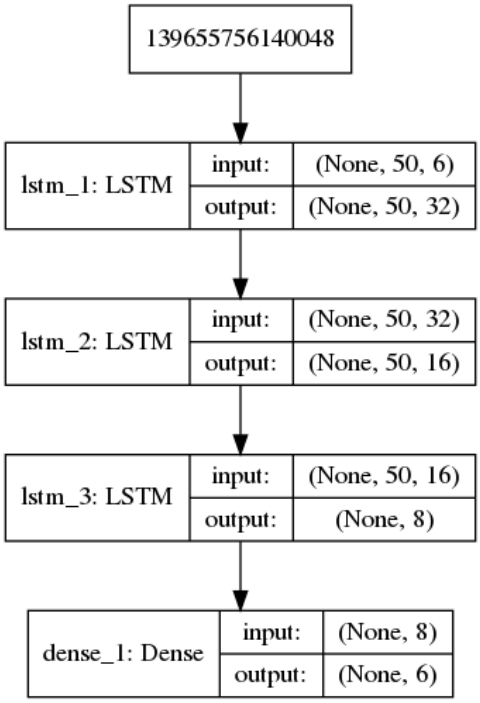
Název modelu	Vrstvy	LTSM	TCN	Dense	Activation	Optimizer	Loss
30mins_50back_250epochs_LSTM_1	4	3		1	relu	adam	mse
30mins_50back_250epochs_LSTM_2	3	2		1	relu	adam	mse
30mins_50back_250epochs_TCN_0	3		1	1		adam	mse
30mins_50back_500epochs_LSTM_0	3	2		1	relu	adam	mse
30mins_50back_500epochs_TCN_0	2		1	1		adam	mse
30mins_50back_50epochs_LSTM_0	4	3		1	relu	adam	mse
30mins_50back_50epochs_LSTM_1	2	1		1	relu	adam	mse
30mins_50back_50epochs_LSTM_2	3	2		1	relu	adam	mse

Název modelu	Hodnocení			
	1. test	2. test	3. test	celkově
30mins_100back_100epochs_LSTM_0	2.55803	1.80608	2.04333	2.13581
30mins_100back_50epochs_LSTM_0	1.94051	1.71778	2.16066	1.93965
30mins_100back_50epochs_LSTM_1	2.47495	2.16184	2.8188	2.4852
30mins_100back_50epochs_TCN_0	2.55068	1.9748	2.32596	2.28381
30mins_50back_100epochs_LSTM_0	2.43125	2.14445	2.36011	2.31193
30mins_50back_100epochs_LSTM_1	1.96385	1.72518	2.30101	1.99668
30mins_50back_100epochs_LSTM_10	1.94459	1.74176	2.31299	1.99978
30mins_50back_100epochs_LSTM_11	1.57483	1.49443	1.71244	1.5939
30mins_50back_100epochs_LSTM_12	1.25145	1.28263	1.2482	1.26076
30mins_50back_100epochs_LSTM_13	2.0503	1.92743	3.25421	2.41065
30mins_50back_100epochs_LSTM_14	1.96008	1.9363	2.82542	2.2406
30mins_50back_100epochs_LSTM_15	2.56119	1.61047	2.72325	2.2983
30mins_50back_100epochs_LSTM_16	1.90659	2.20399	2.83679	2.31579
30mins_50back_100epochs_LSTM_17	2.07165	1.96007	3.34333	2.45835
30mins_50back_100epochs_LSTM_18	1.03849	0.91412	1.06304	1.00522
30mins_50back_100epochs_LSTM_19	2.27788	1.57535	2.51472	2.12265
30mins_50back_100epochs_LSTM_2	2.98089	1.74516	3.97203	2.89936
30mins_50back_100epochs_LSTM_20	3.33238	1.75836	5.54894	3.54656
30mins_50back_100epochs_LSTM_3	2.0404	2.22592	2.52665	2.26432
30mins_50back_100epochs_LSTM_4	2.18439	1.79363	2.89409	2.2907
30mins_50back_100epochs_LSTM_5	2.31986	1.83607	2.98374	2.37989
30mins_50back_100epochs_LSTM_6	2.0217	1.8546	2.26494	2.04708
30mins_50back_100epochs_LSTM_7	1.92271	1.99086	2.57465	2.16274
30mins_50back_100epochs_LSTM_8	2.06293	1.77937	2.37494	2.07241
30mins_50back_100epochs_LSTM_9	1.90253	2.29936	2.24836	2.15008
30mins_50back_100epochs_TCN_0	2.54925	2.13254	3.09537	2.59238
30mins_50back_20epochs_LSTM_0	1.95238	1.74261	2.29591	1.99697
30mins_50back_20epochs_LSTM_1	1.86954	2.12268	2.54735	2.17986
30mins_50back_250epochs_LSTM_0	4.04588	1.62938	4.93793	3.53773
30mins_50back_250epochs_LSTM_1	1.94741	1.75112	2.28628	1.99494
30mins_50back_250epochs_LSTM_2	2.81309	1.63706	3.3617	2.60395
30mins_50back_250epochs_TCN_0	2.74127	1.69265	2.00219	2.14537
30mins_50back_500epochs_LSTM_0	3.18972	1.5067	3.53336	2.74326
30mins_50back_500epochs_TCN_0	2.7142	1.96945	2.26445	2.31604
30mins_50back_50epochs_LSTM_0	2.23505	1.80702	2.71251	2.25152
30mins_50back_50epochs_LSTM_1	3.19133	1.92558	4.7106	3.27584
30mins_50back_50epochs_LSTM_2	3.17962	1.51432	3.57956	2.75783

Tabulka 7: Tabulka výsledků testování

A.2 Popisy

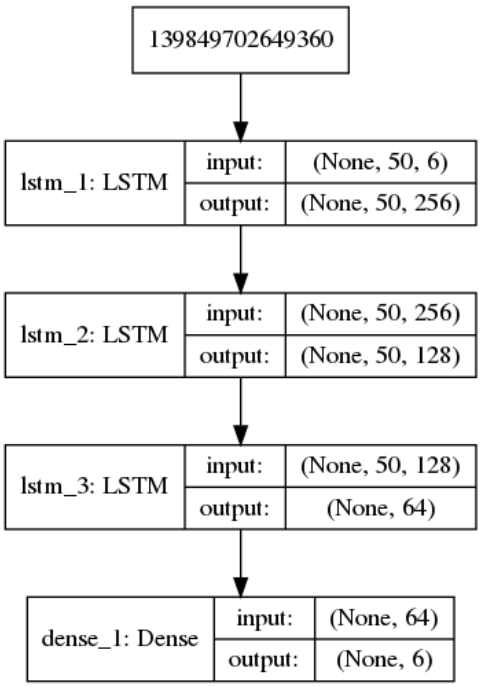
Zde jsou zobrazeny všechny LSTM modely, které jsem uvedl v tabulce 6.



Model sítě

Název
30mins_50back_20epochs_LSTM_0
30mins_50back_20epochs_LSTM_1

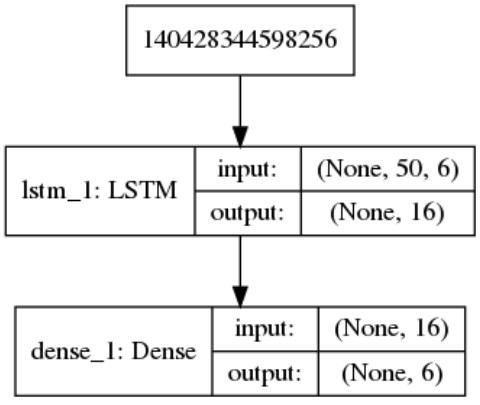
Seznam modelů



Model sítě

Název
30mins_50back_50epochs_LSTM_0
30mins_50back_250epochs_LSTM_1

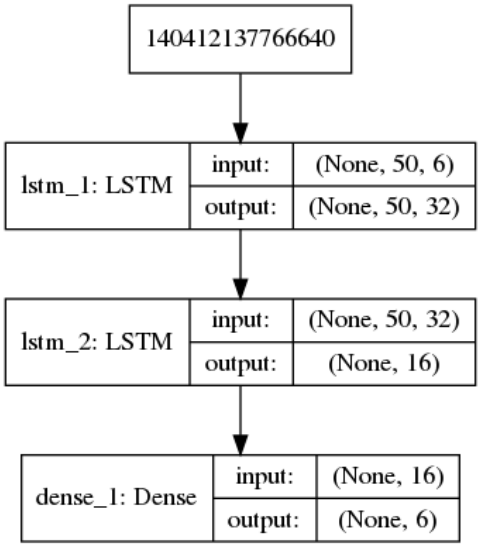
Seznam modelů



Model sítě

Název
30mins_50back_50epochs_LSTM_1

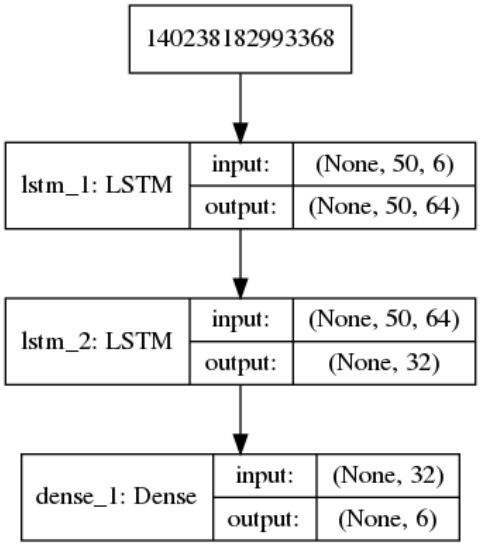
Seznam modelů



Model sítě

Název
30mins_50back_50epochs_LSTM_2

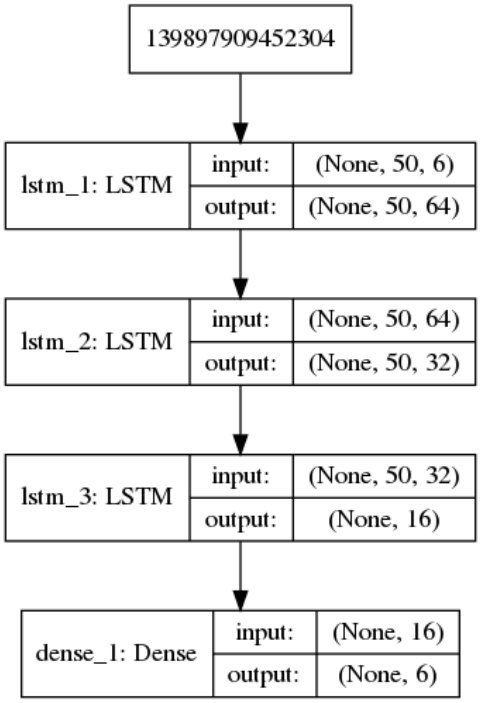
Seznam modelů



Model sítě

Název
30mins_50back_100epochs_LSTM_0

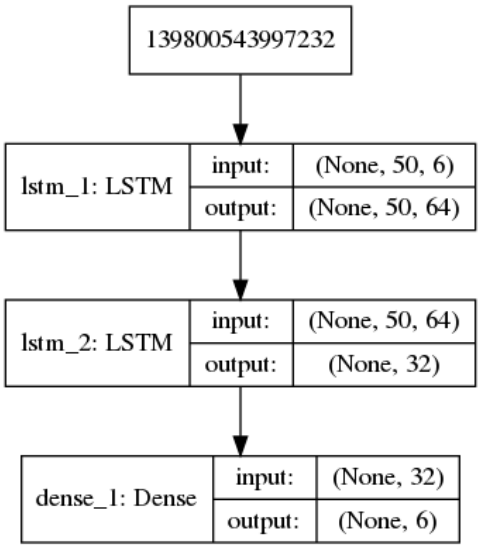
Seznam modelů



Model sítě

Název
30mins_50back_100epochs_LSTM_1
30mins_50back_100epochs_LSTM_2

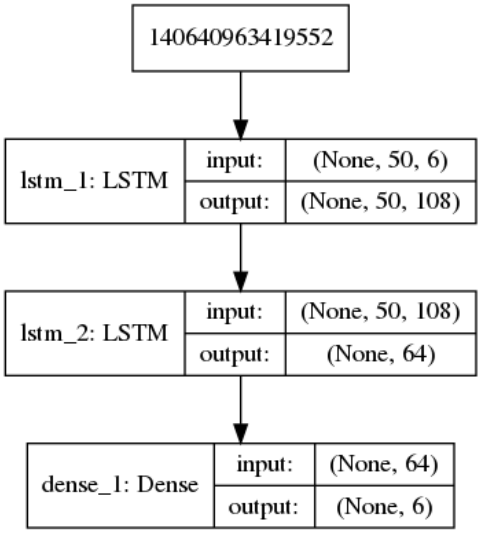
Seznam modelů



Model sítě

Název
30mins_50back_100epochs_LSTM_3
30mins_50back_100epochs_LSTM_4
30mins_50back_100epochs_LSTM_5
30mins_50back_100epochs_LSTM_6
30mins_50back_100epochs_LSTM_7
30mins_50back_100epochs_LSTM_8
30mins_50back_100epochs_LSTM_9
30mins_50back_100epochs_LSTM_10
30mins_50back_100epochs_LSTM_11
30mins_50back_100epochs_LSTM_12
30mins_50back_100epochs_LSTM_15
30mins_50back_100epochs_LSTM_16
30mins_50back_100epochs_LSTM_17
30mins_50back_100epochs_LSTM_18
30mins_50back_100epochs_LSTM_19
30mins_50back_250epochs_LSTM_0
30mins_50back_500epochs_LSTM_0

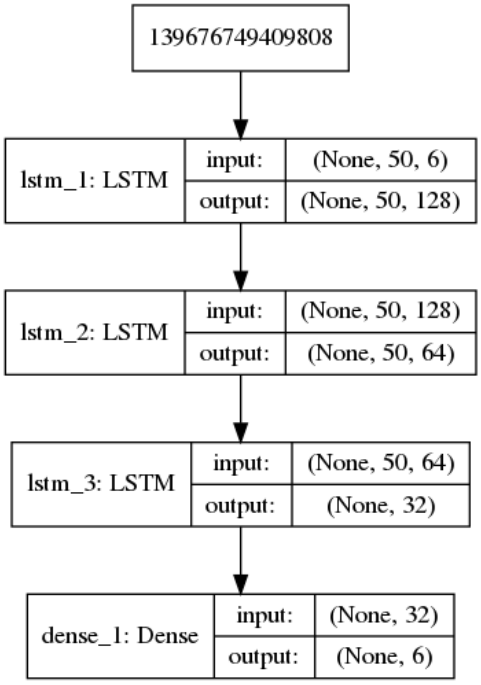
Seznam modelů



Model sítě

Název
30mins_50back_100epochs_LSTM_13

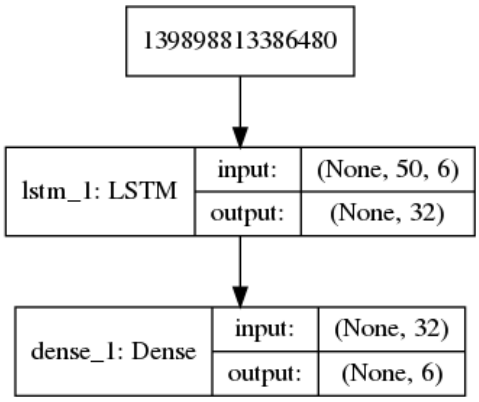
Seznam modelů



Model sítě

Název
30mins_50back_100epochs_LSTM_14

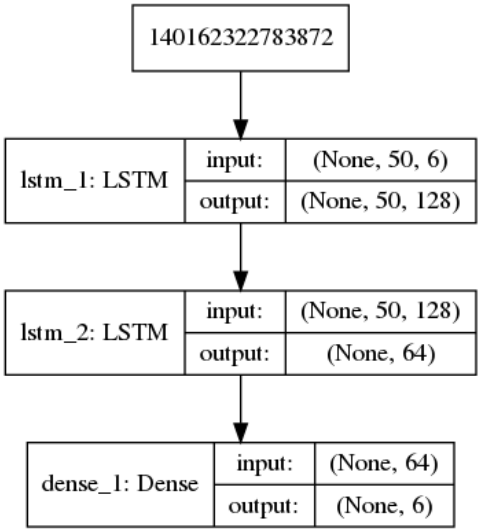
Seznam modelů



Model sítě

Název
30mins_50back_100epochs_LSTM_20

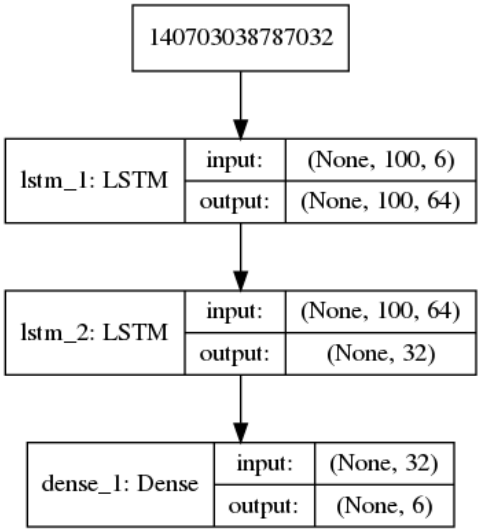
Seznam modelů



Model sítě

Název
30mins_50back_250epochs_LSTM_2

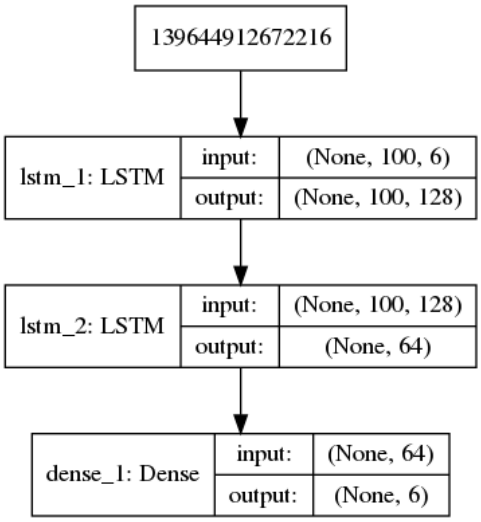
Seznam modelů



Model sítě

Název
30mins_100back_50epochs_LSTM_0

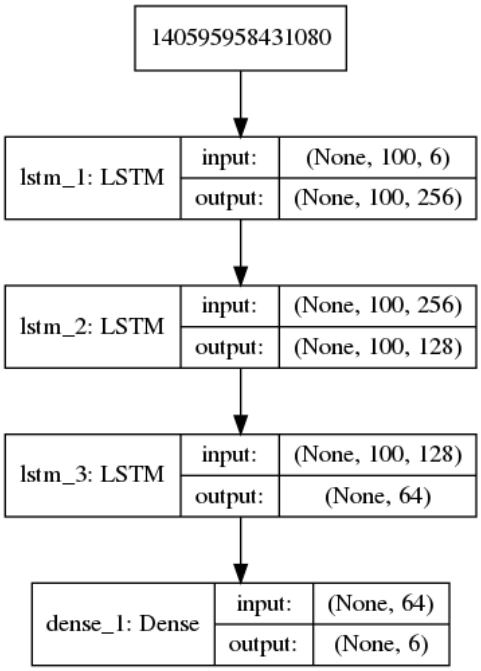
Seznam modelů



Model sítě

Název
30mins_100back_50epochs_LSTM_1

Seznam modelů



Model sítě

Název
30mins_100back_100epochs_LSTM_0

Seznam modelů